Simulation System for Missile Configurations

in

A Distributed Heterogeneous Environment

ARO Proposal No. 33957–MA, Contract No. DAAH04–95–1–0076

Final Report

by

Bharat K. Soni

Professor, Aerospace Engineering

Mississippi State University, MS 39762

Submitted to

U.S. Army Research Office

P.O. Box 12211

Research Triangle Park, NC 27709–2211

DISTRIBUTION UNLIMITED

REPORT DOCUMENTATION PAGE

Form Approved
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1997	Final 1 Jan 95 - 31 Oct 97

4. TITLE AND SUBTITLE

Simulation System for Missile Configurations in a Distributed Heterogeneous Environment

5. FUNDING NUMBERS

DAAH04-95-1-0076

6. AUTHOR(S)

Bharat K. Soni

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES)

Mississippi State University
P.O. Box 9627
Mississippi State, MS 39762

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

ARO 33957.1-MA

11. SUPPLEMENTARY NOTES

The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12 b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

An overall objective of this project is to develop a comprehensive simulation system tailored for missile configurations executable in a distributed heterogeneous environment. The heterogeneous environment will be based on the Network Distributed Global Memory (NDGM) model developed at the Army Research Laboratory (ARL). The integration includes implementation and enhancement of grid generation, flow solution and visualization software modules tailored for missile configurations. The grid generation module will be enhanced for solution adaptivity and for efficient and accurate geometry treatment. This broadbased project will be accomplished in collaboration with researchers at the ARL. The successful development of this system will provide significant productivity improvement associated with simulations involving missile configurations.

14. SUBJECT TERMS	15. NUMBER IF PAGES
	16. PRICE CODE

17. SECURITY CLASSIFICATION OR REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

TABLE OF CONTENTS

APPENDIX

1. INTRODUCTION

Rapid increases in available computational power, as well as advances in algorithm development, have facilitated the analysis of very complicated engineering problems. These engineering problems, however, require various stages of analysis in different engineering disciplines, such as flow field analysis and structural analysis, in order to validate the design. These stages are traditionally performed separately using different software packages. This practice is extremely inefficient and time–consuming and do not contributes the quality of the results. Hence, it would be advantageous to couple these diverse analysis systems into a seamless package, which could be used by a design engineer without specialized training in each specific area.

The calculation of flow fields by means of Computational Fluid Dynamics (CFD) techniques has gained a lot of success in recent years. This is due to the fact that the accuracy and reliability of CFD calculations have been improved dramatically. Currently, it is a common practice to use CFD to help in validating the flow field associated with a new design. There are three major steps for a complete CFD simulation cycle. They are: (i) *grid generation* (pre–processing) (ii) *flow solution calculation* (processing), and (iii) *data visualization and validation* (post–processing). For a complicated geometry, such as a missile with appendages, the grid generation step is usually a labor–intensive process, and it generally takes more than half of the wall–clock time for a complete CFD simulation cycle. TIGER [1, 2] is a recently developed system which drastically speeds the grid generation process for such configurations. The second step, flow solution calculation, however, is a CPU–intensive process. It usually requires more than 80% of the computer time for a typical CFD application. The last step of the CFD cycle is to analyze/validate the data obtained from the second step. This step is neither labor–intensive as the grid generation step, nor as CPU–intensive as the second step.

The processing stage is generally performed on a supercomputer, while the other stages are usually executed on a local graphics workstation. This implies that there are common actions (i.e. data transferal, data manipulation, boundary condition setup, etc.), which a user must perform to use the separate packages. These actions are usually highly tedious and time–consuming. Moreover, the heavily–loaded supercomputers generally have long lists of jobs waiting in the queue, which may result in a long and unbearable turn–around time, not to mention that supercomputers are usually not available to the design engineer in a small to medium size enterprise. Therefore, how to improve the efficiency of engineering applications and how to utilize the available resources become important issues that need to be addressed.

Recent advances in computer hardware and architectures have promoted the development of parallel computing in a distributed, heterogeneous environment. Such an approach allows the decomposition of large engineering problems that require intensive floating point calculations into smaller "blocks", each of which can be handled by a workstation. Engineering problems associated with CFD calculations require intensive floating point calculation. For a detailed flow simulation, it usually requires a supercomputer to calculate the governing partial differential equation, known as the Navier–Stokes equations. By breaking down the domain into smaller blocks and by allowing each block to communicate through a network, we can use a cluster of available workstations to calculate the flow solution, visualize the result, etc. in a distributed sense without using supercomputers. However, this distributed approach requires transferring explicit messages between the cooperating workstations. Keeping track of these messages is usually a difficult and error prone task.

Network Distributed Global Memory, NDGM [3], developed at Army Research Laboratory (ARL) is a shared memory model. NDGM is implemented on top of a message passing layer

called MRS, Message Relay System. It allows an application to copy sections of local memory to and from the global address space. This model also allows users to invoke any other processes which may then access the data from the shared memory.

NDGM has been implemented in the flow solver DZONAL [4], also developed at ARL, along with a simple visualization tool, BopView, in this NDGM environment. This environment allows a user to integrate any number of networked workstations to contribute to the overall memory and share the load of the calculation.

The grid generation code, TIGER [1, 2], is developed at Mississippi State University Engineering Research Center. This code is a customized code for axisymmetric configurations. It is capable of generating grids for turbomachinery and missile configurations in a fraction of the time that is required when using general–purpose grid code.

An overall goal of this project is to establish an environment which can be utilized for a wide spectrum of physical modeling and analysis disciplines. An initial goal is to integrate the key elements of a CFD application (TIGER, DZONAL, and BopView) to build a comprehensive missile simulation system will perform grid generation, flow solution calculation, and solution visualization utilizing the NDGM shared memory, heterogeneous environment. The utilization of a cluster of heterogeneous workstations allows all available resources to be allocated as needed. The cumulative power of such system should be sufficient for even the most computation–intensive problems. Multi–disciplinary analysis capability can be added in a modular fashion into this model as desired. The close coupling of the components and central control by the graphical user interface should provide significant productivity improvement verses current practice.

In a complete CFD cycle, grid generation, flow solution calculation, and solution visualization/ validation are the three major steps. Traditionally, these steps are usually performed independently. therefore, a lot of precious time and effort is wasted due to the improper data transferring, inconsistent data format, and repeatedly tedious solution monitoring during the initial stage of flow parameter setup. Integration of existing softwares as modules into a common environment is a promising approach to reduce the potential for errors in these steps of a CFD application. A comprehensive simulation system should at least include the grid generation module, flow solution calculation module, and visualization module. These modules, though, need to be couples to some degree, and should be independent of one another so that any of them may be replaced in a timely fashion for either performing different tasks, or for simply updating the algorithms.

The grid generation process is currently performed in a user–friendly graphical environment. However, to generate a grid for a missile configuration, it may still take days or even weeks of effort using a general purpose grid generation code, since the user must hand–carve each single boundary and manipulate the surfaces manually. TIGER, a customized grid generation code that is originally developed for turbomachinery applications, is an appropriate choice to be the grid generation module of this integrated missile simulation system since it has been optimized for axisymmetric configurations. It has proved that it reduces the grid generation effort by an order of magnitude for complicated turbomachinery geometries, as compared to the effort required when using a general purpose grid code.

During the initial stage of running the flow code for a configuration, it is traditional for an engineer to run the flow code for a small number of iterations on a supercomputer, transfer the data back to the local host, and invoke a flow solution visualization software to check if the

parameters and boundary conditions are set properly. Such a process may iterate several times before the engineer is confident enough to make a long run of the flow code. This process, however, is tedious not only because the user needs to wait for the queue on the supercomputer, but also because each operation takes time, such as transferring the huge data set back from the remote supercomputer. Using a cluster of workstations on the local network to share the load of such calculation reduces load on the supercomputer and hence speeds the turn around time. Moreover, using the NDGM shared memory scheme, a user not only can distribute the calculation to the cooperating workstations, but also can invoke various processes, such as a visualization module, that accesses the data from the same shared memory as the flow solver. This reduces the effort of transferring the data explicitly through the network.

The proposed system will be built in the shared memory environment, with different softwares in different engineering disciplines being integrated into the system as modules. These modules will be easy to replace by similar softwares for easy update. The shared memory environment should also be studied to improve its efficiency. Software in different disciplines need to be coupled so that the communications of different modules can be smooth and they can be added into the system as desired. Such a system will significantly increase the productivity for solving problems with the utilization of available local resources.

2. RESEARCH OBJECTIVES & MOTIVATION

In a complete CFD (Computational Fluid Dynamics) cycle, geometry–grid generation, execution of flow solver (numerical solution of non linear partial differential equations pertinent to fluid phenomena to be simulated), and solution visualization/validation are the three major steps. Traditionally, these steps are performed independently. Therefore, a significant amount of precious time and effort is wasted due to the improper data transferring, inconsistent data format, and repeatedly tedious solution monitoring during the initial stage of flow parameter setup. Also, geometry–grid generation (pre–processing), and solution visualization (post–processing) are performed on graphical workstations. However, the flow solver is executed on the mainframe or a supercomputer in most cases. Integration of existing software for these functionalities as modules into a common distributed/parallel computing environment is a promising approach to reduce an overall cycle time and the potential errors in the aforestated steps.

An overall objective for this project is to develop a comprehensive simulation system tailored for missile configurations by integrating grid generation, flow solution and visualization as software modules. These modules, though, need to be coupled to some degree, should be independent of one another so that any of them may be replaced (or updated) in a timely fashion. The grid generation module should be enhanced for solution adaptivity and for efficient and accurate geometry treatment.

3. TECHNICAL APPROACH

The development of such an integrated system in distributed computing environment is a formidable task. The approach adapted in this project calls for utilization of existing modules for efficiency. In this regard:

- Grid Generation: The TIGER system developed at the Mississippi State University (MSU) is utilized for grid generation. The TIGER system is a grid generation

software customized for axisymmetric three–dimensional configurations. A detailed description of the TIGER system is provided in Appendix A.1.

- Flow Solver: At initial stage the computer code PARC, developed at Arnold Engineering Development Center, for simulating thin layered Navier Stokes equations is used for flow calculations.
- Solution Visualization: The visualization module BopView of the TIGER system is used as visualization module.
- Distributed Environment: The Network Distributed Global Memory (NDGM) system developed at the Army Research Laboratory is a shared memory model implemented on top of a message passing layer called MRS (Message Relay System). This system is utilized as a computing platform to accomplish distributed processing.
- Geometry and CAD Interface: The computer code CAGI (Computer Aided Grid Interface) developed at MSU is utilized for geometry/surface preparations.
- Adapt3d: The adaptive grid system under development at MSU is used for grid adaption and redistribution.

The development of this integrated system is accomplished in following four phases: (i) POP (Proof of Principle) system – the POP system is accomplished by considering the TIGER system as the basis. The grid generation and flow visualization modules of the TIGER are integrated with the PARC system to simulated missile flow in an integrated fashion. A network module to allow data communication between heterogeneous environment based on TCP/IP communication protocols is developed for data transfer. This system should be considered as a testbed to demonstrate the feasibility of an overall integration system.

(ii) Grid Adaptation and CAGI system: the grid adaptive algorithms and CAGI (Computer Aided Grid Interface) system modules should be enhanced to facilitate geometrical entities encountered in general missile configurations. These modules should be enhanced and prepared for the automatic inclusion in the integrated system.

(iii) Enhancements in Grid adaptive algorithms: the grid adaptive algorithms should be enhanced to improve the efficiency, robustness and accuracy in view of parallel/distributed environment. The grid adaptive module in Adapt3d is based on the grid movement (keep the grid points fixed) pertinent to solution features. The equi–distribution principle is applied in higher dimension to establish distribution criteria. An appropriate blending of algebraic and elliptic systems are utilized to accomplished well distributed, smooth and near orthogonal adaptive grid. The enhancements must address parallelization of the multiblock elliptic solver utilized in the grid adaptive system.

(iv) CFD–NDGM System: the final phase is to bring all modules together and incorporate them in the NDGM environment. This work is to be accomplished in collaboration with ARL researchers.

4. SIGNIFICANT ACCOMPLISHMENTS

The first three phases of this integration project have been accomplished. However, the last phase has not been accomplished at the level where all the modules operate in the CFD–NDGM environment very coherently.

The integrated testbed system using TIGER code has been completed and validated utilizing missile configurations of interest to Army Research Laboratory (ARL). The pre and post processing steps associated with the integrated system can be executed on an SGI workstation. However, the flow solver can be executed on an SGI Workstation. The system has been successfully implemented in the existing NDGM System at ARL. A five block grid configuration associated with the Wrap around missile and the resulting execution speed on the power challenge array obtained by ARL researchers are presented in figure 1–2. The CFD simulation, in parallel, associated with multiple duct engine configurations were performed with the integrated TIGER System. The geometrical configuration along with the speed–up obtained are presented in the figures 3–4. The Tiger code description is provided in Appendix A.

The NonUniform Rational BSpline (NURBS) based grid generation technology has been enhanced to improve grid quality in view of optimizing smoothness and orthogonality by redistributing interior NURBS control points. A fast NURBS curve–surface evaluation algorithm developed, under NASA/MSFC contract has been incorporated for evaluation of qarametric NURB entities.

The detailed technical discussions on this development is reported in the following publication:

- Shih, M.H., Stokes, M.L., Huddleston, D. and Soni, B.K., "Towards in integrated CFD System on a Parallel Environment," Parallel Computational Fluid Dynamics: Implementations and Results Using Parallel Computers, A. Eccer, J. Periaux, N. Satofuka and S. Taylor (Eds.), pp. 437–445, AIAA–95–2338, 1995.

This publication is enclosed in the technical publication section 5.1.

The grid adaptive and CAGI (Computer Aided Grid Interface) modules have been enhanced to efficiently address missile configurations. All functional modules are now ready to be implemented on the overall NDGM framework. This work is being accomplished by working side–by–side with ARL researchers. The detailed results and the testbed integrated system has been transferred to ARL.

New weight functions employing boolean sums is developed to represent local truncation errors for adaptation. The adaptive procedure redistributes mesh points based on existing flow fields. The technique has been applied to a missile configuration (associated with KTA2–12 project) where flow field involves supersonic freestream conditions at angle of attack, large scale separated vortical flow, vartex–vortex and vartex–surface interactions separated shear layers and multiple shocks of different intensity. The results have demonstrated the capability of adaptive aforestated complicated flow feating.

The technical discussions associated with this development reported in:

- Thornburg, H., Soni, B.K. and Boyalakuntla, K., "A Structured Based Solution–Adaptive Technique for Complex Separated Flows," Journal of Applied Mathematics & Computation, to appear.

This paper is enclosed in section 5.2.

Wrap Around Fin

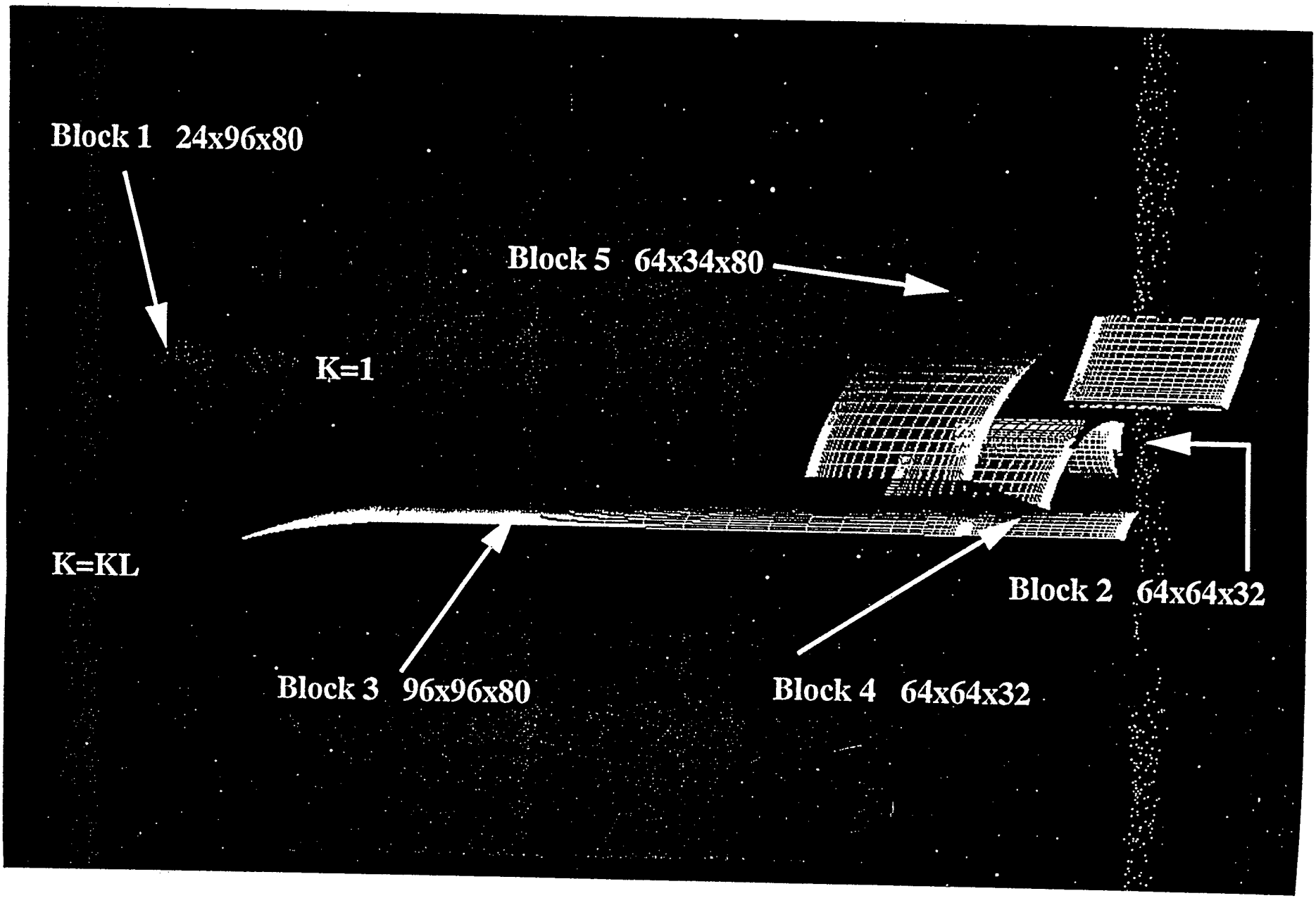


Figure 1. Fire Block Grid-Wrap Around Missile

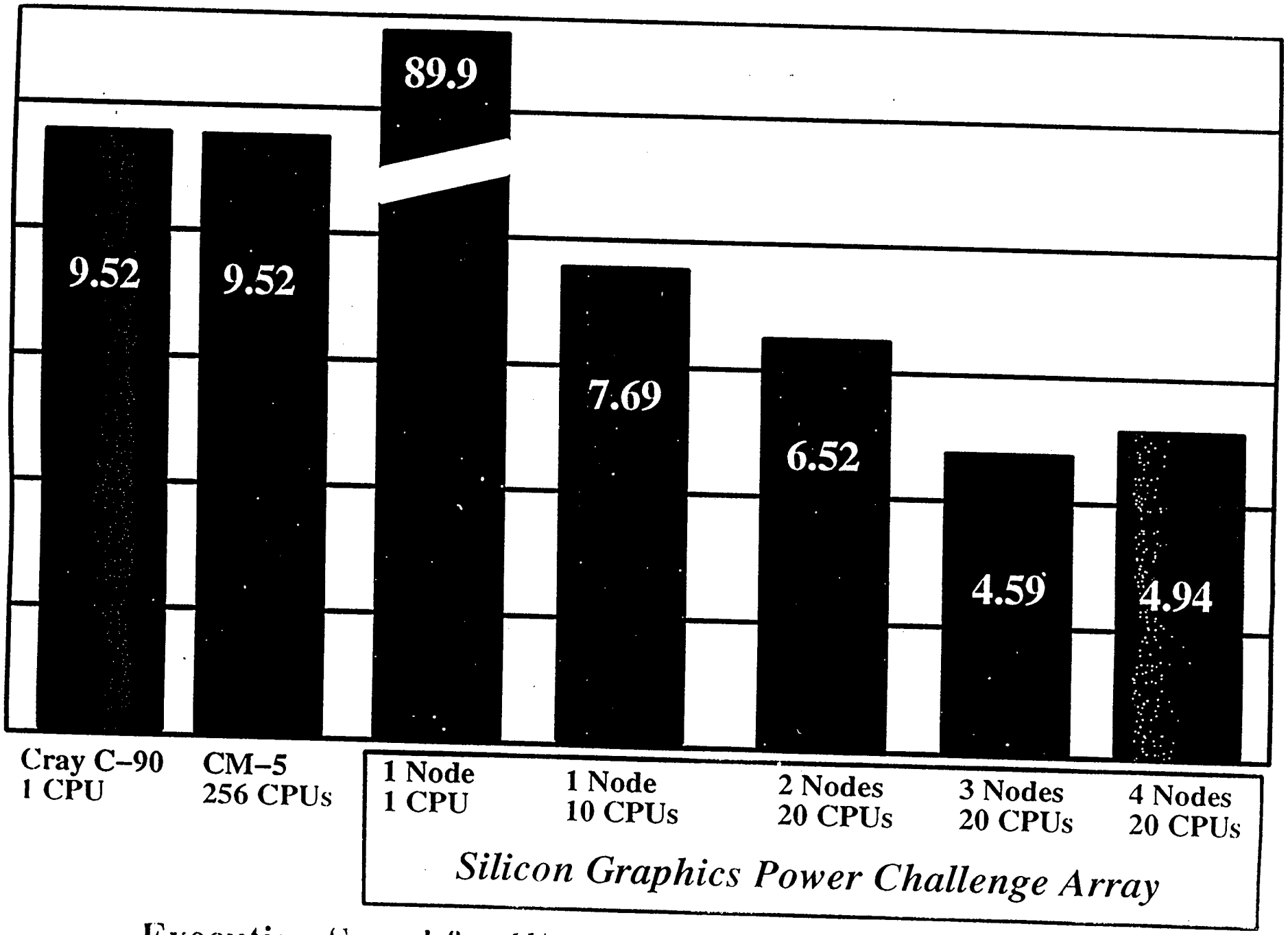


Figure 2. Execution Speed for Wrap–Around Fin Configuration

Seconds / Iteration

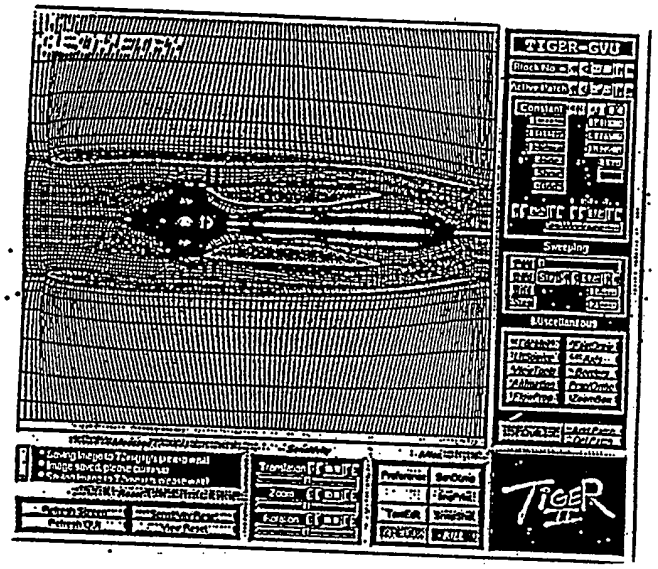


Fig.3 Multiple Duct Engine Configuration

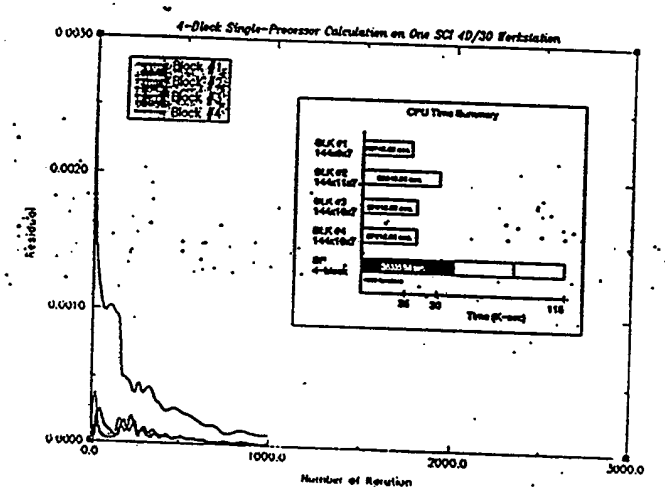


Fig.4 Convergence History

A Parallel Multiblock Eliptic Grid Generator for structured grids in any complex topology has been designed. The features of this code are:

- Each block is run on an individual processor with load balancing done by threads on multi–processor shared memory machines.
- Code designed to have minimum cache misses for efficient operation on shared memory machines.
- MPI used as the message passing library.
- Parallel I/O as far as possible.
- Solution based adaptive control of grids
- Movement of body surface points using NURBS.

Tests on a 5 block grid (~1.3 million points) around a missile fin have shown speedup of 6.72 times using 7 processors of an 8 processor. This configuration is presented in figure. 5.

SGI–ONYX

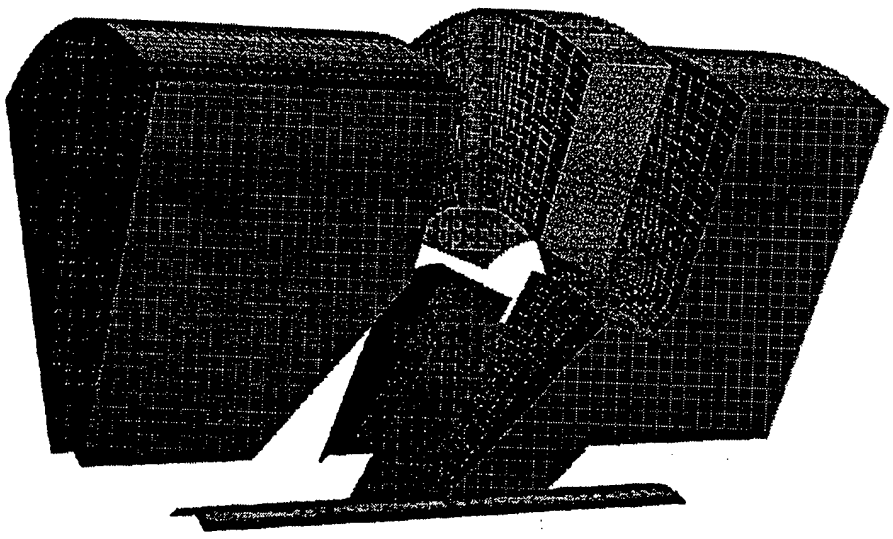


Figure 5. Multiblock Grid Around Missile–Fin.

A detailed technical discussions on this development are reported in the following two publications.

- "Parallel Solution–Adaptive structured Grid Generator for complex multiblock topologies", Proceedings of the International conference on parallel and distributed processing techniques and applications, 1997.
- "Parallel Adaptive Grid Generation for structured multiblock domains" Master's thesis, Mississippi State University, May, 1997.

These publications are included in section 5.3 and 5.4 respectively.

All these modules have been transferred to the researchers at ARL. These modules will be incorporated in the NDGM–DICE environment at ARL. The successful development of this system will provide significant productivity improvement associated with CFD simulation involving complex configurations.

5.0 Technical Publications

5.1 "Towards An Integrated CFS System in a parallel environment", AIAA–95–2338

Parallel Computational Fluid Dynamics:
Implementations and Results Using Parallel Computers
A. Ecer, J. Periaux, N. Satofuka and S. Taylor (Editors)

Towards an Integrated CFD System in a Parallel Environment

M. Shih*,
M. Stokes+,
D. Huddleston!,
B. Soni@

ABSTRACT

This paper describes the strategies applied in parallelization of CFD solvers in a parallel distributed memory environment using a message passing paradigm. To monitor the simulation process, indirect rendering via the OpenGL graphics API is investigated and shown to exihibit excellent performance. A customized, integrated CFD simulation system, TIGER, tailored for turbomachinery configurations, executable in a distributed environment is developed. computational examples demonstrating the successful parallelization of the NPARC code with real−time visualization and the TIGER system are presented. Future expansion to this environment is discussed.

1. INTRODUCTION

The primary objective of the National Science Foundation's Engineering Research Center (ERC) at Mississippi State University is to reduce the man/machine resource required to perform Computational Field Simulation(CFS). The approach to reducing the machine resource is two−fold: 1) design of specialized hardware such as multi−computers with very low latency communications, and 2) the design of distributed computing environments to take advantage of heterogeneous collections of personal computers, workstations, and supercomputers if available. The approach to reducing the manpower resource is primarily through the development of advanced software tools which efficiently generate and edit grids, and visualize the solution field.

Classically, the process of CFS is thought of as a pre−processing phase of grid generation, the solution phase, and a post−processing phase comprised of visualization. However, examination of this process in a production environment illustrates these phases are not separate at all, and typically exist in at least binary pairs. For example, grid generation requires visualization of the grids during the construction process to be effective. Grid generation requires one or more iterations of the field solution to adequately capture viscous phenomena and shocks. The solution process also requires visualization to determine the accuracy of boundary and run−time parameters. For more complicated problems, the solution phase may require an integrated grid generation capability to provide solution adaptivity.

To address this problem, the ERC formed the Integration Thrust to investigate the concept of integrated environments for the CFS process. Conceptually, an integrated environment would reduce or simplify the logistics of moving between phases and therefore reduce wall−clock time as well as

*. Post Doctorate, NSF Engineering Research Center for Computational Field Simulation
+. Research Faculty, NSF Engineering Research Center for Computational Field Simulation
!. Associate Professor, Civil Engineering, Mississippi State University
@ Professor, Aerospace Engineering, Mississippi State University

reduce the expertise required to perform simulations. Early testbeds for integrated systems revealed that simple–minded integration of existing grid generation, solver, and visualization tools were impractical, and would have limited longevity due the volatile nature of current day tools. These testbeds resulted in two conclusions which coincide with near–term and far–term goals, respectively:

1. The solver set–up/execution phase and real–time visualization can be integrated in a distributed environment using *off–the–shelf* technology.

2. Grid generation codes must be redesigned to support distributed memory environments and support object–oriented concepts and therefore are not currently mature enough for integration.

Concentration in this paper is placed upon the solver set–up/executation phase and the real–time visualization.

The casual observer may wonder why visualization must be distributed and why it should be real–time. The anwer to the first question is that current trends in CFS point to using a distributed memory parallel environment for large scale applications[1.]. With simulations routinely containing more than a million nodes, memory requirements for graphic workstations would overwhelm even the most current hardware, thus the distributed architecture is required to mass enough memory to contain the problem. The real–time features are required to monitor early or transient phases of the solution to minimize loss of computing resource due to incorrect boundary or run–time parameters. Note that what is needed here is not publication quality graphics, but rather graphics which illustrate faults in the solution process.

The progress realized in the development of the aforestated integrated environments for the CFS process is presented in this paper. In particular, the development associated with following issues are described:

i. Parallelizing the widely utilized off–the–shelf CFD system. The NPARC code[2.] was chosen for this research effort due to it's availability to US research and academic communities, and that it is heavily used in US Federal Labs and industry. Though the NPARC code has been parallelized by private companies such as Boeing[3.], no parallel version was available for general distribution as of the time of this writting. The NPARC code was a natural choose for parallelization because of its multi–block structured design. This aspect is discussed in detail in the following section.

ii. Real–time visualization in a a distributed environment. Here, an architecture for indirect rendering using OpenGL graphics library is described

iii. Customized integrated CFD simulation process for a class of configurations. An integrated system, TIGER, customized for turbomachinery application is described.

2. THE NPARC SOLVER

The NPARC solver started life as a derivative of the ARC[7] suite of codes from NASA Ames, but gained popularity after being heavily modified by Cooper[2] at Arnold Engineering Development Center(AEDC) in Tullahoma, TN. under the name PARC, or Propulsion ARC code. The NPARC code, a close derivative of PARC, solves either the Reynolds–averaged Navier–Stokes equations or the Euler equations, subject to user specification. NPARC uses a standard central difference algorithm in flux calculations, with Jameson[9] style artificial dissipation applied to maintain stability. NPARC has two–time integration algorithms available including Pulliam's[7] diagonalized Beam and Warming approximate factorization algorithm or a multi–stage Runge–Kutta integration. The diagonalized algorithm was applied herein. Among other characteristics, NPARC utilizes conservative metric differencing, offers local time–step options, and provides a generalized boundary treatment. The code can be applied in either a single or multi–block mode. This results in a code which is quite general and easy to apply to complex configurations. The NPARC multi–block code supports only one grid block in memory at a time, caching the other blocks to disk. The main loop would write out the current

block (if not the first time), read in and process the second block, and the cycle would repeat. Similarily, boundary conditions were handled by files which were subsequently read by adjacent blocks once in memory. This design made it quite simple to produce a parallel code from this original design.

The PANARC, or Parallel National ARC code, as coined at the ERC, took the original design and actuall simplified it. Block caching was eliminated such that each process only operated on one block, and the boundary condition routines replaced the previous I/O calls with PVM[10] send/receive calls. Boundary and run−time conditions are defined in a FORTRAN namelist file and were not altered from the original format. However, each process only reads what is pertinent for each respective block.

To simplify the run−time logistics, PANARC assumes each process can access the same set of input files, which under the UNIX Operating System, is accomplished by NFS. PVM was choosen for the interprocess communication library, over the then immature MPI[11] libary because MPI lacks support for dynamic process management. However, it is apparent that from commercial industry support among a large class of workstation and supercomputer vendors, MPI will be the preferred choice for the message passing interface of the future. Features lacking now are expected to be included in future versions of MPI.

Memory in the NPARC code is allocated in the main program as a single work array. The only change to support indirect rendering was to replace the DIMENSION statement with a C language call that allocates shared memory (see Figure 1). This change allows any other process (with proper permissions) to access the main memory used by the solver. It is through this mechanism that the visualization module gains access to the field variables.

3. ARCHITECTURE FOR INDIRECT RENDERING USING OPENGL

The premise for the design for this system is the concept of domain decomposition, which relies on the solution domain being broken down into (for this case) equal size domains, each domain being mapped to a remote computer. To minimize data transfer during the visualization phase, the rendering calls are issued on local data rather than transfering the data to the display host (which was not an option since the graphics workstation did not have enough memory to store the data in the first place). This concept relies on the Method of Extracts[12] which suggests that what is being visualized is a subset or *extraction* dimensionally lower than that of the original data. In terms of solution monitoring, this is almost always the case.

A design contraint was for the visualization scheme be as unobtrusive as possible to the simulation process as well as minimize code modification to the solver. In addition, the visualization capability must be attachable/detachable at any time during the simulation. The process toplogy that evolved is shown in Figure 1.

In this topology, visualization is facilitated as a separate process on each host. Communication between the solver and visualization process on each processor is through shared memory. Optional features in this topology(shown in dashed lines) are (1) the existence of a solver process on the display host (not recommended for large scale visualization), and (2) enabled message passing between a visualization client on the display host and the solver clients. The latter option may be convenient, but generally is not recommended in that it violates the requirement that the solver and visualization processes be independent.

In the current model, the visualization client running on the display host has the responsibity of creating the shared window, acting on mouse and button events(which are not shared), clearing the various buffers before rendering begins, setting the projection matrix, and synchronizing the swapping of the back display buffer. It may also have the responsibility of rendering information for a domain if the user has elected to use the display host as an active node in the simulation. The visualization clients on the remote processors render to the shared window on the display host through the multi−threaded X11 Server, not the visualization client running on the display host. This greatly simplifies the programming aspects of distributed applications.

Communications between the visualization clients are primarily one−way, that is flowing toward the display hosts. The OpenGL[] calls are tokenized on the remote clients and sent to the X11 Server running on the display host using the GLX extension provided by Silicon Graphics. Thus, the

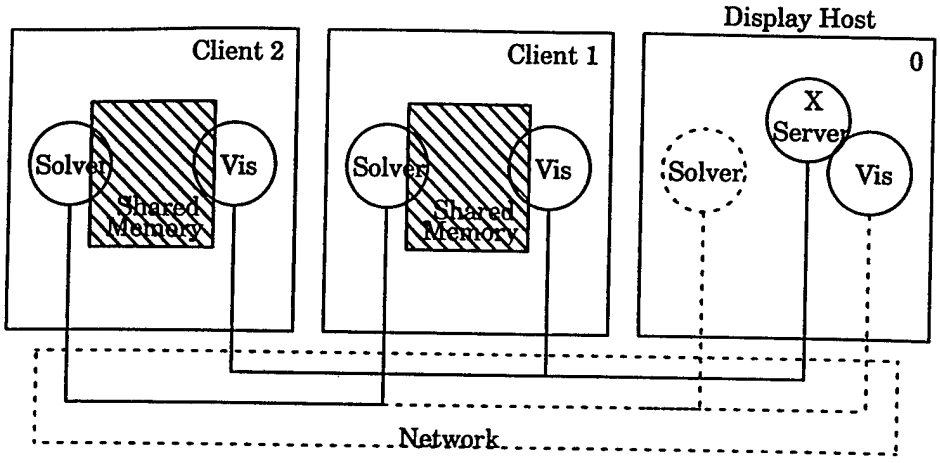


Figure 1. Process Topology of Indirect Rendering

programmer can think of indirect rendering simply as providing addition input streams into the geometry engines on the display host. Since this operation is often performed unsynchonized, the scan conversion may not be unique since these tokens may arrive unordered to the server. This is not generally a problem, because most operations are not dependent on the ordering. However, if alpha buffering(for example) is enabled with background blending, then the ordering of the operations are important and care must be taken.

It is often necessary to communicate information from the display host to the remote clients. Three different methods are described herein with the choice of methods dependent on the needs of the application. If the information is global in nature, then X Windows provides the mechanism known as properties, which are shared among local and remote processes sharing a common window. If any client attempts to change a global property, then a *PropertyChangeEvent* is sent to all clients, which respond with a request to read the new property and thus synchronize the data. If point–to–point communication is required, then a simple method is to post a window (such as Motif or Xt window) on the display processor where the user can use graphical techniques such as buttons or sliders to convey information. A third technique is to use client defined *XSendEvents* that support the passing of arbitrary information between local or remote processes. Care must be exercised when exchanging data between heterogenous machines, as data incompatibilities may arise. This problem can be avoided if the data is encoded/decoded using the publicly available XDR routines.

When double buffered graphics are required on the display host, it is the responsibility of the owner of the window(the visualization process on the display host) to actually swap the display buffers after the remote clients have completed their rendering operations. To perform this operation, the local client must have two pieces of information: (1) the number of remote clients sharing the window, and (2) notification when all clients have completed their operation. To satisfy (1), the visualization clients simply send a *ClientMessage* telling the client on the display host that a new remote client is registering itself. The local client simply keeps track of the number of registered clients. This information coupled with the XSynchronization Extension to X Windows allows the client on the display host to maintain a synchronized window.

4. TIGER SYSTEM

The development of TIGER has evolved from a grid generation system into an integrated system specialized in applications in rotating turbo–machinery. The integrated system is comprised of 6 modules: (*i*) grid generation module, (*ii*) visualization module, (*iii*) network module, (*iv*) flow solution module, (*v*) simulation module, and (*vi*) toolbox module. Each module may be accessed indepen-

dently to perform different tasks. These modules are linked together with a common graphical user interface (GUI).

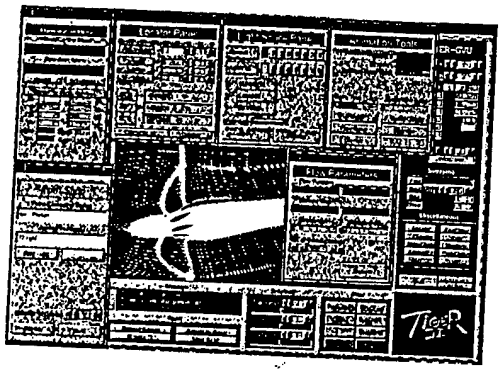


Figure 2. Graphical User Interface of TIGER

TIGER's GUI, as shown in Figure 2, was developed using the FORMS Library[14]. The interface contains a global panel window, an image display window, and various sub−panels and auxiliary panels. The global panel, where the logo, message browser, image controls widgets, and other global function buttons reside, is the main panel of the entire GUI. The image display window sits in the left−hand upper corner of the global panel and is the window that allows the user to view the grid and the solution. Sub−panels such as the "GVU" panel, pop up on the right hand portion of the global panel as the algorithm requests user inputs. These panels provide the necessary widgets for a particular group of user inputs. Auxiliary panels, such as the "Animation Tools" and "Flow Property" panels, pop up only upon the user's request for parameter customization. A help window, available upon request, consists of a large browser window to display the contents of the help files, and several buttons for the user to select topics.

The simulation capability in TIGER system is comprised of the network module, visualization module, and the flow solution module. As shown in Figure 3, the setup of this module is to use the network module as the data bridge that conveys user's commands and the flow solution between the local graphics workstation and a remote super computer. It consists of two sub−modules: the *client−server* sub−module, which resides in TIGER as a function, and the *server−client* sub−module which is a set of routines that reside independently on the remote mainframe. The network setup in the system allows two−way communication between the local workstation and the remote supercomputer, which allows the flow data to be sent back to local machine for the update. Figure 3 represents the setup of the network module. The setup of the simulation module is illustrated in Figure 4.

The flow solver that resides on the super computer may be any appropriate software. Currently, a multi−stage unsteady turbomachinery code (MSUTC)[15] is adopted as the flow solution module. This code applies the thin−layer approximation with a Baldwin−Lomax turbulence model to solve the Reynolds−averaged Navier−Stokes equations. It is an implicit finite volume scheme with flux Jacobians evaluated by flux−vector−splitting and residual flux by Roe's flux−difference−splitting. A modified two−pass matrix solver based on the Gauss−Siedel iteration was used to replace the standard LU factorization to enhance the code's stability. This code, uses localized grid distortion technique [15−16] to the buffer zone between the blade rows to achieve time−accurate solution for compressible flow. It performs flow calculation on the supercomputer, and outputs the flow solution every interval, whose length is specified by the user. The evolving solution is automatically updated on the screen graphically through the execution of visualization module. The visualization module is designed to allow scientific visualization for both the grid and flow solutions with basic rendering

methods, such as wireframe and Gouraud shading, contour and vector field. Time–dependent grid and/or solution can be animated dynamically.

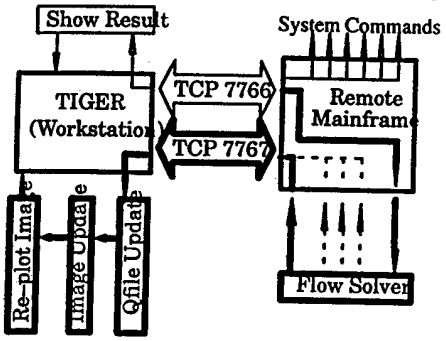


Figure 3. Tiger Network Topology

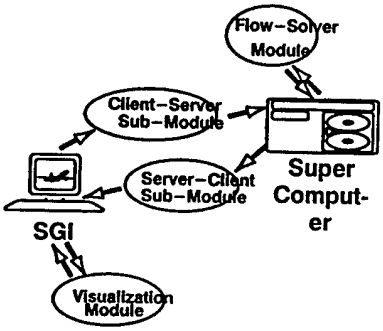


Figure 4. Simulation Module Setup

5. COMPUTATIONAL EXAMPLES

To test the design of the integration of the distributed parallel environment with distributed visualization, a four block 3D test case was performed using a four block 16mb SGI Indigo Elans workstations on a 10 mb Ethernet network(Figure 5 shows the block configuration on the vertical symmetry plane). The test case is an internal engine configuration with blades removed. Figure 6 is a six frame sequence of pressure on a vertical symmetry plane as captured directly from the screen.

For the given test cases, the latency created by network traffic is almost negligible. To further minimize thie effect, the X server on the display host is capable of caching display lists created on either the remote or local clients. The impact of caching is that objects that have not changed are not retransferred across the network, thus local redraws even for complex objects are very fast.

For the given test cases, the latency created by network traffic is almost negligible. To further minimize this effect, the X server on the display host is capable of caching display lists created on either the remote or local clients . The impact of this caching is that objects that have not changed are not retransfered across the network, thus local redraws even for complex objects are very fast.

Hamilton–Standard's single rotation propfan SR–7 is presented as an example simulation application. This geometry is modeled using a C–type domain with a grid size of 31x16x9, as illustrated in Figure 7. This geometry has 8 blades and the blade stagger angle is set to be 54.97 degrees, with an advance ratio J=3.07 (rpm=1840) and a free–stream Mach number 0.78. Each iteration of an Euler solution will take about 5 seconds on the aforementioned SGI Challenge machine. Density contours of the 2000th iteration is shown in Figure 8.

6. FUTURE WORK

The current effort clearly illustrates the need for an integrated desktop tool for the execution and field monitoring of CFS. This desktop, while generally not adding any new capability, aids the user by either automating functions, or simplifying the use of the tools by bringing them together in a common Graphical User Interface(GUI). A few of these tools are listed below.

Graphical based input/run parameters–
Currently, NPARC reads a FORTRAN namelist input file for information regarding the run–time and boundary conditions. A valuable aid to the user would be to allow these parameters to be edited graphically, and then have the GUI regenerate the namelist input files

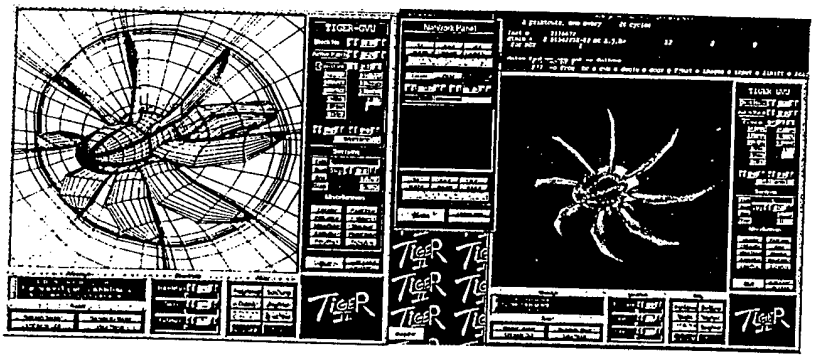


Figure 7. Grid for SR−7

Figure 8. Density contour for iteration 2000

for the user. The GUI would allow the user to indicate boundary conditions by *point−and−clip* thereby combining the editing process with graphical validation of the input.

An automatic mesh decomposition tool−
This feature, given a set of rules customizable by the user, would automatically decompose an arbitrary set of structured grid blocks into the number of equal sized grid blocks comensurate with the number of available processors in the distributed environment. This tool, given the connectivity information of the blocks, would be capable of breaking structured grids into sub−grids, including those that cross existing grid boundaries assuming the grid interfaces are point matched. This tool would also be capable of automatically calculating the boundary conditions on sub−grids.

Visualization monitoring tool−
This would allow the user to graphically monitor the state of a running simulation using the standard graphical features available in other post−processing packages such as line or garound shaded field properties, iso−surfaces, vector plots[5], streamlines, and particle paths[4.] to name a few. The interface would provide feature detection such as monitoring of flow angles on slip surfaces, or massive defects in total temperature or pressure, to name two. The ability to monitor global as well as local grid convergence at a glance would provide instant information on the *health* of the simulation.

Performance monitoring of the distributed parallel system−
This feature would allow the user to monitor the performance of the distributed system. Many of these packages already exist[5. ,6.], and would therefore only require merging into the integrated system.

As we look forward into the future of CFS as defined by the requirements for multi−disciplinary optimization and analysis(MDO&A), one can define requirements that must be present in grid generation systems to provide adequate functionality in an integrated system. These include

1. Ability to self−adapt to changes in either geometry or topology. Most current grid programs either don't support editing, or do it very poorly.
2. Closely related to 1, grid components must remember and maintain design constraints during the editing process. Without default editing methods, 1 would be undefined.
3. The inputs in the editing process must not be required to have intimate knowledge of the editing process itself. For example, if the location of a bezier control point attached to a line is altered and that line is connected to the edge of a surface, the procedure that requests the point to move need not be required to specify the series of steps necessary to reconstruct the surface. That information should be associated with the surface itself.

4. The editing process must not be linked uniquely to the GUI. Anyone or any process should be able to execute an edit, not just the indivual with the mouse.
5. Edits should occur in parallel.
6. Grids should be constructed in a collaborative environment. Many individuals should be able to work on components of a complicated grid simultaneously.
7. Grids components should be constructed relative to arbitrary defined sub−ordinate coordinate systems which are free to relocate in 3 space. This construct would be an extension of a grouping operator in which the geometic operators of translation, scaling, and rotation are defined.

8. REFERENCES

[1] Taylor, Steve, and Wang, Johnson, "Large−scale Simulation of the Delta II Launch Vehicle," *proceedings of Parallel CFD 95*, California Institute of Technology, June 26−28, 1995.

[2] Cooper, G.K., and Sirbough, J.R., "PARC Code: Theory and Usage," AEDC−TR−89.15, Arnold Engineering Development Center, Arnold AFB, 1989.

[3] Lewis, Jeffrey, "Domain−Decomposition Parallelization of the PARC Navier−Stokes Code," *proceedings of Parallel CFD 95*, California Institute of Technology, June 26−28, 1995.

[4] SHIH, M.H., "*TIGER: Turbomachinery Interactive Grid genERation*," Master's Thesis, Mississippi State University, December 1989.

[5] SHIH, M.H., "Towards a Comprehensive Computational Simulation System for Turbomachinery," Ph.D. Dissertation, Mississippi State University, May 1994.

[6] SHIH, M.H., YU, T−Y, and SONI, B.K., "Interactive Grid Generation and NURBS Applications", Accepted for print on Journal of Applied Mathematics and Computations, vol. 65:1−2, pp. 345−354, 1994.

[7] Pulliam, J.H., "Euler and Thin Layer Vanier−Stokes Codes: ARC2D, ARC3D", *Notes for Computational Fluid Dynamics User's Workshop*, The University of Tennessee Space Institute, Tullahoma, TN, UTSI Publication E02−4005−023−84, p. 151, March 1984.

[8] Cooper, G.R. and Sirbaugh, J.R., "The PARC Distinction: A Practical Flow Simulator", AIAA−90−2002, *AIAA/ASME/SAE/ASEE 26th Joint Propulsion Conference*, Orlando, FL, July 1990.

[9] Jameson, A., Schmidt, N. and Turkel, E., "Numerical Solutions of the Euler Equations by Finite Volume Methods Using Runge−Kutta Time−Stepping Schemer", AIAA−81−1259, *AIAA 14th Fluid and Plasma Dynamics Conference*, Palo Alto, CA, 1981.

[10] PVM reference: http://www.netlib.org/pvm3/

[11] MPI reference: http://www.mcs.anl.gov/mpi/

[12] STOKES, M.L., HUDDLESTON, D.H., and REMOTIQUE, M.G., "*A Proactical Model for Multidisciplinary Analysis Data and Algorithm*," 6th SIAM Conference on Parallel Processing for Scientific Computing, Norfolk, VA, March 1993.

[13] SONI, B.K., THOMPSON, J. F., STOKES, M.L., and SHIH, M. H., "*GENIE^{++}, EAGLEVIWE and TIGER: General Purpose and Special Purpose Graphically Interactive Grid Systems*," AIAA−92−0071, AIAA 30th Aerospace Sciences Meeting, Reno, NV, January 1992.

[14] OVERMARS, M.H., "*Forms Library: A Graphical User Interface Toolkit for Silicon Graphics Workstations*," Version 2.1, Utrecht, Netherland, November 1992.

[15] CHEN, J.P. and WHITFIELD, D.L., *"Navier–Stokes Calculations for The Unsteady Flowfiled of Turbomachinery,"* AIAA–93–0676, AIAA 31st Aerospace Sciences Meeting, Reno, NV, January 1993.

[16] JANUS, J.M., *"Advanced 3–D CFD Algorithm for Turbomachinery,"* Ph.D. Dissertation, Mississippi State University, Mississippi, May 1989.

5.2 Structured Grid Based Solution–Adaptive Technique for complex separated flows", Journal of Applied Mathematics and Computation, to appear.

"Structured Grid Based Solution–Adaptive
Technique for complex separated flows",
Journal of Applied Mathematics and Computation, to appear.

A Structured Grid Based Solution–Adaptive Technique for Complex Separated Flows

by

Hugh Thornburg, Bharat K. Soni and Boyalakuntla Kishore
NSF Engineering Research Center for
Computational Field Simulation
Mississippi State University
Mississippi State, MS 39762

ABSTRACT

A structured grid based technique is presented to enhance the predictive capability of widely used CFD codes through the use of solution adaptive gridding. The procedure redistributes mesh points based upon an existing flowfield. A newly developed weight function employing Boolean sums is utilized to represent the local truncation error. This weight function is then used to construct forcing functions associated with an elliptic system. A NURBS representation is employed to define block surfaces for boundary point redistribution. The technique has been applied to a flowfield about a configuration of practical interest. This flowfield involves supersonic freestream conditions at angle of attack, and exhibits large scale separated vortical flow, vortex–vortex and vortex–surface interactions, separated shear layers and multiple shocks of different intensity. The results demonstrate the capability of the developed weight function to detect shocks of differing strengths, primary and secondary vortices, and shear layers adequately.

INTRODUCTION

Rapid access to highly accurate data about complex configurations is needed for multi–disciplinary optimization and design. In order to efficiently meet these requirements a closer coupling between the analysis algorithms and the discretization process is needed. In some cases, such as free surface, temporally varying geometries, and fluid structure interaction, the need is unavoidable. In other cases the need is to rapidly generate and modify high quality grids. Techniques such as unstructured and/or solution–adaptive methods can be used to speed the grid generation process and to automatically cluster mesh points in regions of interest. Global features of the flow can be significantly affected by isolated regions of inadequately resolved flow. These regions may not exhibit high gradients and can be difficult to detect. Thus excessive resolution in certain regions does not necessarily increase the accuracy of the overall solution.

Several approaches have been employed for both structured and unstructured grid adaption. The most widely used involve grid point redistribution, local grid point enrichment/derefinement or local modification of the actual flow solver. However, the success of any one of these methods ultimately depends on the feature detection algorithm used to determine solution domain regions which require a fine mesh for their accurate representation. Typically, weight functions are constructed to mimic the local truncation error and may require substantial user input. Most problems of engineering interest involve multi–block grids and widely disparate length scales. Hence, it is desirable that the adaptive grid feature detection algorithm be developed to recognize flow structures of different type as well as differing intensity, and adequately address scaling and normalization across blocks.

These weight functions can then be used to construct blending functions for algebraic redistribution, interpolation functions for unstructured grid generation, forcing functions to attract/repel points in an elliptic system, or to trigger local refinement, based upon application of an equidistribution principle. The popularity of solution-adaptive techniques is growing in tandem with unstructured methods. The difficultly of precisely controlling mesh densities and orientations with current unstructured grid generation systems has driven the use of solution-adaptive meshing Use of derivatives of density or pressure are widely used for construction of such weight functions, and have been proven very successful for inviscid flows with shocks[2,7,11]. However, less success has been realized for flowfields with viscous layers, vortices or shocks of disparate strength. It is difficult to maintain the appropriate mesh point spacing in the various regions which require a fine spacing for adequate resolution. Mesh points often migrate from important regions due to refinement of dominant features. An example of this is the well know tendency of adaptive methods to increase the resolution of shocks in the flowfield around airfoils, but in the incorrect location due to inadequate resolution of the stagnation region. This problem has been the motivation for this research.

The weight functions developed utilize scaled derivatives and normalizing procedures to minimize or eliminate the need for user input. The most attractive feature of this work is the ability to detect flow features of varying intensity and the lack of user defined inputs for the selection of the weight function.

In this research a NURBS representation is employed to define block surfaces for boundary point redistribution. The features described have been implemented into Adapt2D/3D. An adaptive grid system capable of automatically resolving complex flows with shock waves, expansion waves, shear layers and complex vortex-vortex and vortex-surface interactions. An adaptive grid approach seems well suited for such problems in which the spatial distribution of length scales is not known a priori.

APPROACH TO ADAPTION

The elliptic generation system:

$$\sum_{i=1}^{3}\sum_{j=1}^{3} g^{ij}\hat{r}_{\xi^i\xi^j} + \sum_{k=1}^{3} g^{kk}P_k\hat{r}_{\xi^k} = 0 \qquad (1)$$

where
- r : Position vector,
- g^{ij} : Contravariant metric tensor
- ξ^i : Curvilinear coordinate, and
- P_k : Control function.

is widely used for grid generation [1]. Control of the distribution and characteristics of a grid system can be achieved by varying the values of the control functions P_k in Equation 1 [1]. The application of the one dimensional form of Equation 1 combined with equidistribution of the weight function results in the definition of a set of control functions for three dimensions.

$$P_i = \frac{(W_i)_{\xi^i}}{W_i} \quad (i = 1,2,3) \qquad (2)$$

These control functions were generalized by Eiseman [2] as:

$$P_i = \sum_{j=1}^{3} \frac{g^{ij}(W_i)_{\xi^j}}{g^{ii}\ W_i} \quad (i = 1,2,3) \qquad (3)$$

In order to conserve some of the geometrical characteristics of the original grid the definition of the control functions is extended as:

$$P_i = \left(P_{initial_{geometry}}\right) + c_i(P_{wt}) \quad (i = 1,2,3) \tag{4}$$

where $P_{initial\ geometry}$: Control function based on initial geometry

P_{wt} : Control function based on current solution

c_i : Constant weight factor.

These control functions are evaluated based on the current grid at the adaption step. This can be formulated as:

$$P_i^{(n)} = P_i^{(n-1)} + c_i(P_{wt})^{(n-1)} \quad (i = 1,2,3) \tag{5}$$

where

$$P_i^{(1)} = P_i^{(0)} + c_i(P_{wt})^{(0)} \quad (i = 1,2,3) \tag{6}$$

A flow solution is first obtained with an initial grid. Then the control functions P_i are evaluated in accordance with Equations 2 and 5, which is based on a combination of the geometry of the current grid and the weight functions associated with the current flow solution[11].

Evaluation of the forcing functions corresponding to the grid input into the adaptation program has proven to be troublesome. Direct solution of Equation 1 for the forcing functions using the input grid coordinates via Cramer's rule or IMSL libraries was not successful. For some grids with very high aspect ratio cells and/or very rapid changes in cell size, the forcing functions became very large. The use of any differencing scheme other than the one used to evaluate the metrics, such as the hybrid upwind scheme[8], would result in very large mesh point movements. An alternative technique for evaluating the forcing functions based on derivatives of the metrics was implemented[3].

$$P_i = \frac{1}{2}\frac{(g_{11})_{\xi^i\xi^i}}{g_{11}} + \frac{1}{2}\frac{(g_{22})_{\xi^i\xi^i}}{g_{22}} + \frac{1}{2}\frac{(g_{33})_{\xi^i\xi^i}}{g_{33}} \quad (i = 1,2,3) \tag{7}$$

This technique has proven to be somewhat more robust, but research efforts are continuing in this area.

SOLUTION PROCEDURE

The software developed inputs two PLOT3D format files [4], one for the grid and one for the flow solution. These files contain the number of blocks, the grid size, the grid coordinates and the solution vector. Output consists of an NPARC[5] restart file, as well as two PLOT3D files of the adapted grid and the flow solution interpolated onto the new grid. A multi-block treatment capability is included. The adaptive grid is constructed in three steps. The first step is to generate the weight functions, which due to their critical importance will be discussed in detail in a separate section. The second step is to generate the actual adapted grid by equidistribution of the aforementioned weight function. In the current work this is accomplished by the numerical solution of Equation 1. A coupled three-dimensional strongly implicit procedure (CSIP) as described by Ghia et al. [6] has been implemented for the solution of the discretized equations. Upwind differencing, with biasing based on the sign of the forcing functions, as well as central differencing has been implemented and studied for the first derivative terms. The first order upwind differencing increases the stability of

the procedure, but at the expense of smearing the grid clustering. Hence, a hybrid upwind/central differencing scheme has been implemented to lessen this smearing while maintaining the smoothness and stability of the upwind procedure. Central differencing has been employed for all second and mixed derivative terms. All non-linear terms were treated by quasi-linearization. Since Equation 1 is solved iteratively, the forcing functions must be evaluated for each new successive grid point location. The forcing functions are obtained by interpolation from the original grid. The interpolation procedure employed was that used for the non-matching block to block interface capability of the NAPRC [5] code. This scheme is based on trilinear interpolation. Upon convergence of this process the flow solution is interpolated onto the adapted grid using the same interpolation subroutine. Calculations can then be continued from the new restart file. This procedure can then be repeated until an acceptable solution is obtained. Experience indicates that coupling this procedure with a code capable of treating time accurate grid movement would ease this process and lessen the CPU requirements.

Currently, grid adaptation is performed in an uncoupled manner. For simple problems this is adequate as only marginal changes are observed after two to three cycles of adaptation. However, for complex flow fields it appears necessary to closely couple the flow solver and the adaptation technique as significant changes in both grid and flow solution are visible after even six cycles of uncoupled adaption. Also, the more complex flows require the use of a hybrid differencing scheme for the solution of the grid equations. For flows containing strong shocks and shear layers along with weaker structures, central differencing was unstable and first order upwind differencing of the grid equations smeared the weaker structures. Therefore to obtain full benefit of the adaptive grid blended central/upwind has been implemented for the grid equations.

WEIGHT FUNCTIONS

Application of the equidistribution law results in grid spacing inversely proportional to the weight function, and hence, the weight function determines the grid point distribution. Ideally, the weight would be the local truncation error ensuring a uniform distribution of error. However, evaluation of the higher-order derivatives contained in the truncation error from the available discrete data is progressively less accurate as the order increases and is subject to noise. Determination of this function is one of the most challenging areas of adaptive grid generation. Lower-order derivatives must be non-zero in regions of wide variation of higher-order derivatives, and are proportional to the rate of variation. Therefore, lower-order derivatives are often used to construct a weight function as a proxy for the truncation error. Construction of these weight functions often requires the user to specify which derivatives and in what proportion they are to be used. This can be a time consuming process. Also, due to the disparate strength of flow features, important features can be lost in the noise of dominant features. The weight functions developed by Soni and Yang [7] and Thornburg and Soni [8] are examples of such efforts. The weight function of Thornburg and Soni [8] has the attractive feature of requiring no user specified inputs. Relative derivatives are used to detect features of varying intensity, so that weaker, but important structures such as vortices are accurately reflected in the weight function. In addition, each conservative flow variable is scaled independently. One-sided differences are used at boundaries, and no-slip boundaries require special treatment since the velocity is zero. This case is handled in the same manner as zero velocity regions in the field. A small value, epsilon in equation 8, is added to all normalizing quantities. In the present work this weight function has been modified using the Boolean sum construction method of Soni [7]. Also, several enhancements of an implementation nature have been employed. For example epsilon has been placed outside the absolute value operator. This eliminated the possibility of spurious gradients in the weight function in regions where epsilon was nearly equal and opposite in sign to the local

normalizing flow variable. Also, the normalizing derivatives have been set to an initial or minimum value of ten percent of the freestream quantities. This alleviates problems encountered in flows without significant features to trigger adaption in one or more coordinate directions. Otherwise a few percent variation would be normalized to the same level as a shock or other strong feature. The current weight function is as follows:

$$W = \frac{W^1}{max(W^1, W^2, W^3)} \oplus \frac{W^2}{max(W^1, W^2, W^3)} \oplus \frac{W^3}{max(W^1, W^2, W^3)}$$

Where, $k=1,2,3,$ and

$$W^k = 1 + \frac{\frac{|\varrho_{\xi^k}|}{|\varrho|+\varepsilon}}{\left(\frac{|\varrho_{\xi^k}|}{|\varrho|+\varepsilon}\right)_{max}} \oplus \frac{\frac{|(\varrho u)_{\xi^k}|}{|(\varrho u)|+\varepsilon}}{\left(\frac{|(\varrho u)_{\xi^k}|}{|(\varrho u)|+\varepsilon}\right)_{max}} \oplus \frac{\frac{|(\varrho v)_{\xi^k}|}{|(\varrho v)|+\varepsilon}}{\left(\frac{|(\varrho v)_{\xi^k}|}{|(\varrho v)|+\varepsilon}\right)_{max}} \oplus \frac{\frac{|(\varrho w)_{\xi^k}|}{|(\varrho w)|+\varepsilon}}{\left(\frac{|(\varrho w)_{\xi^k}|}{|(\varrho w)|+\varepsilon}\right)_{max}}$$

$$\oplus \frac{\frac{|\varrho_{\xi^k\xi^k}|}{|\varrho|+\varepsilon}}{\left(\frac{|\varrho_{\xi^k\xi^k}|}{|\varrho|+\varepsilon}\right)_{max}} \oplus \frac{\frac{|(\varrho u)_{\xi^k\xi^k}|}{|(\varrho u)|+\varepsilon}}{\left(\frac{|(\varrho u)_{\xi^k\xi^k}|}{|(\varrho u)|+\varepsilon}\right)_{max}} \oplus \frac{\frac{|(\varrho v)_{\xi^k\xi^k}|}{|(\varrho v)|+\varepsilon}}{\left(\frac{|(\varrho v)_{\xi^k\xi^k}|}{|(\varrho v)|+\varepsilon}\right)_{max}} \oplus \frac{\frac{|(\varrho w)_{\xi^k\xi^k}|}{|(\varrho w)|+\varepsilon}}{\left(\frac{|(\varrho w)_{\xi^k\xi^k}|}{|(\varrho w)|+\varepsilon}\right)_{max}} \qquad (8)$$

The symbol $\oplus$ represents the Boolean sum. Note that the directional weight functions are scaled using a common maximum in order to maintain the relative strength. For the results shown, the weight function used is the sum of the weight function defined by Eq. 8 and the one defined in [8]. The weight function defined in [8] is Eq. 8 evaluated without using the Boolean sum.

Flow past a wedge at Mach=2.0 is used to illustrate the enhanced detection capabilities of this newly developed weight function. Figure 1 presents weight functions evaluated using the previous procedure, lower half plane, as well as the current procedure, upper half plane.

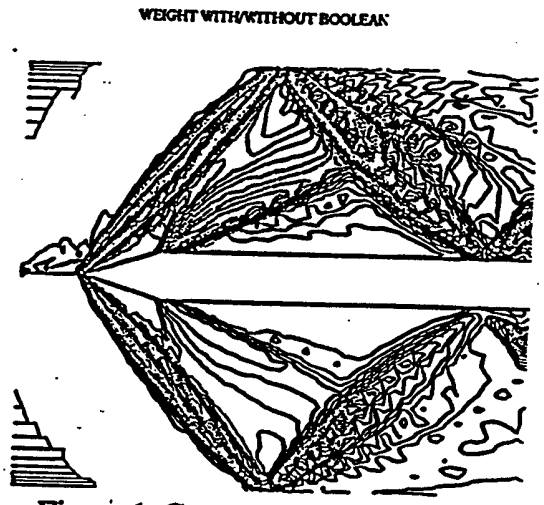


Figure 1. Comparison of Weight Functions.

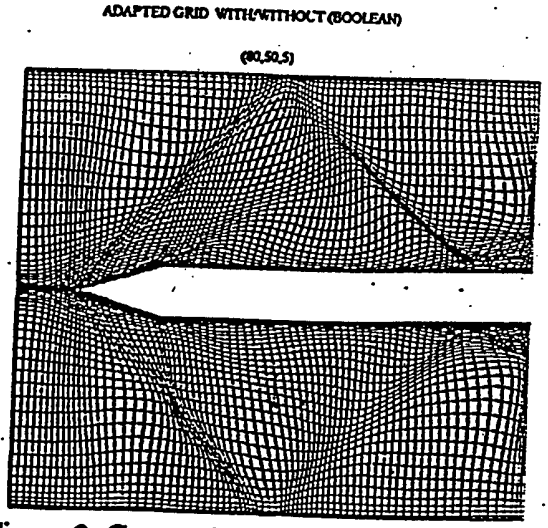


Figure 2. Comparison of Adapted Grids.

It can be observed that both weight functions clearly detected the primary shock. It can also be seen that the expansion fan, boundary layer, and the reflected shocks are much more clearly represented in the current weight function. Adapted grids using both weight function formulations are presented in Fig. 2. The high gradient regions of the expansion region are only reflected in the adapted grid using the new weight function. The reflected shock is also much sharper. Figure 3 compares the solution obtained using the current adaption procedure with that obtained using the original grid. The enhanced resolution is clearly evident.

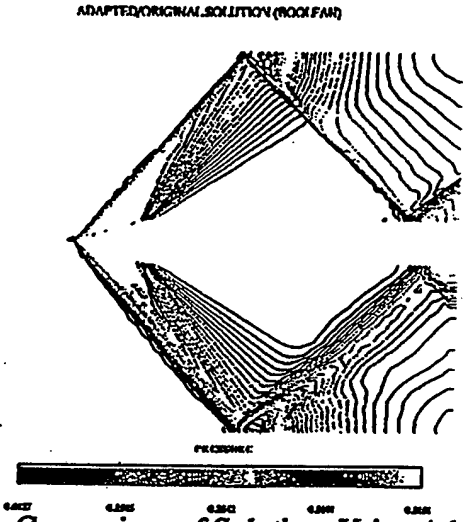


Figure 3. Comparison of Solutions Using Adapted Grid.

BOUNDARY POINT REDISTRIBUTION

Accurate representation of the flowfield in the vicinity of boundaries is critical for an acceptable overall solution. Physics occurring near boundaries often drive the flow physics occurring in other regions of the flowfield. This is especially true for noslip surfaces. Hence, the quality and distribution of the grid in this region is of critical importance. For the implementation of many turbulence models orthogonality is also required. When using an adaptive procedure based on a redistribution of mesh points, such as in this work, the interior points move as the grid is adapted. This leads to distorted cells if the boundary points are not redistributed as the grid is adapted. Both grid quality and geometric fidelity must be maintained during redistribution. In the current work all surfaces of individual blocks are treated in the same manner, whether block interfaces or flow boundary conditions. Two steps must be performed. First the geometry is defined, and secondly the surface mesh is regenerated using a given distribution mesh. The geometry is defined as a NURB surface from the current surface mesh to be redistributed. This procedure is based on the algorithms developed by Yu and Soni for Genie++[9]. This definition is then used to generate the surface using a user specified distribution mesh. The entire surface or a subregion can be redistributed. Subregions can be used to fix points, such as sharp corners or a transition point between boundary condition type. For example block boundary to noslip surface. The distribution mesh is a [0,1] square evaluated from a specified surface based upon arc length. For solid surfaces the distribution mesh is based on the nearest interior coordinate surfaces. The spacing between surfaces is small and the surfaces are of a similar geometric shape. Therefore the normal coordinate is nearly orthogonal to the surface. Block interfaces are treated by redistributing the current block surface based on its corresponding surface in the neighboring block.

FLOW AROUND GENERIC MISSILE CONFIGURATION

Supersonic flow at Mach=1.45 and 14 degree angle of attack has been simulated around a tangent-ogive cylinder. The grid constructed after two adaption cycles using hybrid differencing of the grid equations and the current weight functions is presented is presented in Figure 4.

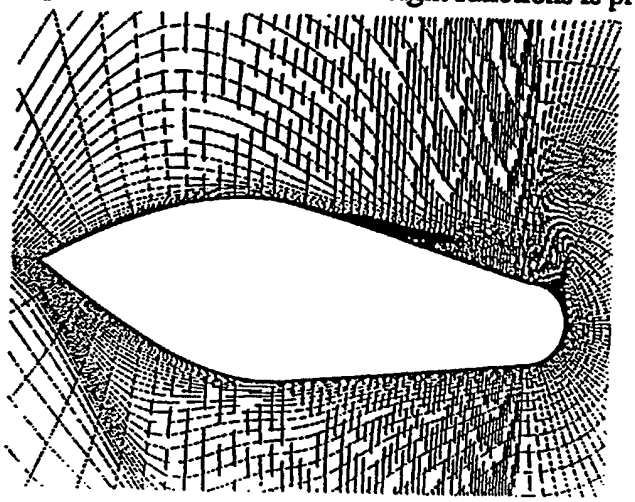


Figure 4. Adapted grid after two cycles.

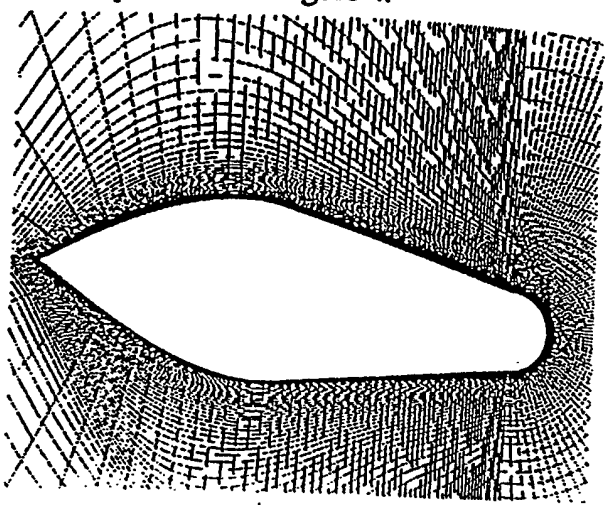


Figure 5. Adapted grid after two cycles.

Figure 5 presents the grid constructed using the previous weight function and the same flow conditions and number of adaptation cycles. Figures 5 and 6 present streamwise cuts of the two grids shown in Figs 4 and 5 at X/D = 5.5 and 7.5 respectively. The left had side of Figures 6 and 7 correspond to Figure 5 and the right hand side corresponds to Figure 5.

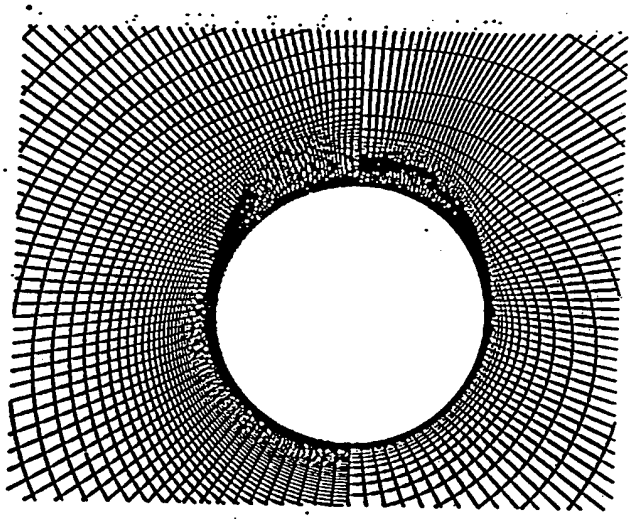


Figure 6. X/D = 5.5

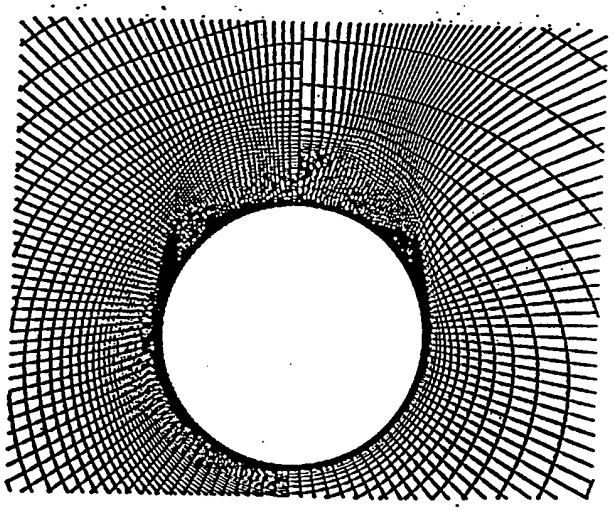


Figure 7. X/D = 8.5.

As can be seen from Figures 4–7 the feeding sheet, and both primary and secondary vortices, as well as the strong shock at the nose are clearly visible. Comparison of Figures 4 and 5 show the sharper resolution provided by the new formulation of the weight function. The shock itself is more sharply represented, but the greatest improvement is in representing the vortex and the shear layer. The previous formulation simply clustered mesh points in the vicinity of the vortex. This does improve resolution. However, the new formulation clearly shows the circular shape of the vortex, as well as the

high gradient regions on the edges of the structure. Hence, the new method of grid adaptation more closely reproduces the flow physics. Also, stronger and sharper concentration of mesh points is observed with the new formulation. This is most apparent in the far field region and the greater detail of the flowfield visible in the grid. Both the primary and secondary separation points are well represented. On the windward side of the missile it can be seen that the boundary layer clustering is greatly increased by the adaptation procedure. This is desirable in this area as the boundary layer is the only feature of interest. Examination of Fig 4 reveals that toward the aft end of the missile, the off surface structure is not well defined. This is because after two cycles of adaptation the structure is not well enough defined by the flow solver and associated grid for the adaptation procedure to sharply define the structure. Only structures that have been at least partially resolved by the flow solver can be detected by the weight function. Hence the quality of the weight function is dependent upon the quality of the solution. It should be noted that resolution of the flowfield improves with each cycle of adaptation. The adaption procedure and the flow solver should be coupled so that the adapted grid can reflect all the features that are detected as the solution progresses and improves due to adaption.

Figure 8 presents the flow solution obtained using the NPARC [5] flow solver, and the KE turbulence model option. Figure 9 presents the associated weight function.

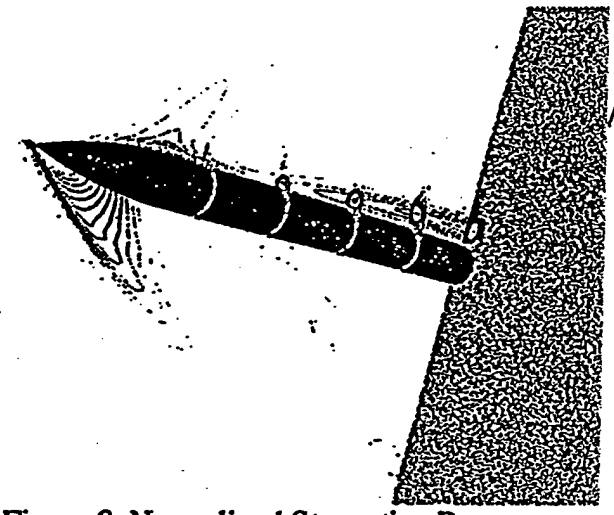


Figure 8. Normalized Stagnation Pressure.

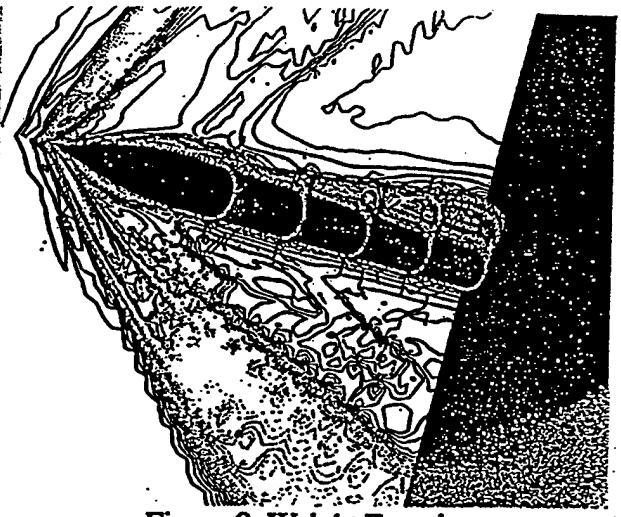


Figure 9. Weight Function.

CONCLUSIONS

The results shown demonstrate the capability of the developed weight function to detect shocks of differing strengths, primary and secondary vortices, and shear layers adequately. For the test case presented the increased resolution of the shock and the expansion region resulted in the structure increasing in size so that interaction with the farfield occurred. Thus, the farfield boundaries should be placed at a larger distance from the body. No user input is required. It is imperative that the adaptation process be coupled with the flow solver as experience indicates that the complex flowfields may require many adaptive cycles. This is particularly critical for flow fields that are not well represented by the initial solution. The adapted grids allow the use of larger time steps to increase the convergence rate. However, the single greatest benefit resulting from the adapted grid is the lowering of the artificial viscosity required for stability. The adapted grid aligns and clusters near shocks and shear layers. For the test problem a dramatic decrease in the artificial viscosity coefficients is

possible when running on the adapted grid. This results in off surface structures which are much sharper and more closely resemble the flow visualization. The boundary layer on the windward side also became thinner and the normal grid spacing was decreased by the adaptive procedure.

Currently multi-block problems are being simulated to evaluate this capability. Attention is being placed upon across block scaling for the weight functions. A global maximum across all blocks seems to work well for the normalization of weight function components. However, this issue is being monitored as more complex problems are attempted. A further problem arises due to equidistribution via forcing functions. This does not appear to be a problem where a block face is connected to one and only one face of another block. Problems have been encountered for a multi-block launch vehicle computation where a block face consisted of a block to block connection and a solid surface for the base region. A smooth, continuous set of forcing functions across this block boundary does not yield a smooth grid. It is believed that this is due to the elliptic character of the grid equations. The simple fix is to introduce artificial block boundaries to force one to one correspondence. Work is continuing on these issues as well as coupling with a flow solver, unsteady flows, and more complex multi-block problems. These results will be published at a later date.

References

1. Thompson, J.F., Warsi, Z.U.A. and Mastin, C.W., Numerical Grid Generation: Foundations and Applications, North-Holland, Amsterdam. 1985.
2. Eiseman, P.R., "Alternating Direction Adaptive Grid Generation," AIAA Paper 83-1937, 1983.
3. Soni, B.K., "Structured Grid Generation in Computational Fluid Dynamics," Vichnevetsky, R., Knight, D., and Richter, G. (eds.), Advances in Computer Methods for Partial Differential Equations VII, Rutgers University, pp 689-695, June 1992.
4. Buning, P.G. and Steger, J.L., "Graphics and Flow Visualization in Computational Fluid Dynamics," AIAA Paper 85-1507-CP, Proceedings of the AIAA 7th Computational Fluid Dynamics Conference, 1985.
5. NASA LeRC and USAF AEDC, "NPARC 1.0 User Notes," June 1993.
6. Ghia, K.N., Ghia, U., Shin, C.T. and Reddy, D.R., "Multigrid Simulation of Asymptotic Curved-Duct Flows Using a Semi-Implicit Numerical Technique," Computers in Flow Prediction and Fluid Dynamics Experiments, ASME Publication, New York 1981.
7. Soni, B.K. and Yang, J.C., "General Purpose Adaptive Grid Generation System," AIAA-92-0664, 30th Aerospace Sciences Meeting, Reno, NV, Jan. 6-9, 1992.
8. Thornburg, H.J. and Soni, B.K., "Weight Functions inGrid Adaption," Proceedings of the 4th International Conference in Numerical Grid Generation in Computational Fluid Dynamics and Related Fields held at Swansea, Wales 6-8th April 1994.
9. Soni, B.K., Thompson, J.F., Stokes, M.L. and Shih, M.H., "GENIE++, EAGLEView and TIGER: General and Special Purpose Interactive Grid Systems," AIAA-92-0071, 30th Aerospace Sciences Meeting, Reno, NV, Jan. 6-9, 1992.
10. NASA LaRC, "User Document for CFL3D/CFL3DE (Version 1.0)", 1993.
11. Thompson, J.F., "A Survey of Dynamically-Adaptive Grids in Numerical Solution of Partial Differential Equations," Applied Numerical Mathematics, vol. 1, pp 3-27, 1985.

5.3 "Parallel Solution–Adaptive Structured Grid Generator for complex multiblock Topologies", Proceedings of the International conference on parallel and distributed processing techniques and applications, June 1997.

Solution–Adaptive Structured Grid Generator
plex multiblock Topologies", Proceedings of the
tional conference on parallel and distributed
sing techniques and applications, June 1997.

Parallel Solution–Adaptive Structured Grid Generator for Complex Multiblock Topologies

Manoj Apte
Computational Engineering
Mississippi State University
Mississippi State, MS 39759,USA

Bharat K. Soni
Computational Engineering
Mississippi State University
Mississippi State, MS 39759,USA

This paper presents a parallel structured grid adaptation algorithm for general three dimensional multiblock domains (PMAG). Grid blocks are computed in individual processes that maybe distributed over multiple processors. MPI is used for message passing. On shared memory machines, each block can be split over multiple threads. The grids are redistributed as a solution of the elliptic partial differential equations. Weight function is computed as a boolean sum of the scaled gradients and curvatures of all flow variables. Neumann boundary condition has been implemented using NURBS to maintain geometric fidelity. Surface definitions are generated by using inverse NURBS formulation. A parallel multiblock solution interpolation algorithm has been incorporated to guarantee accurate adaptation. The algorithm can also be used as a multiblock elliptic grid generator.

1. Introduction

Grid generation is the starting point of any computational fluid dynamics simulation. Creation of a suitable grid over a complex multiblock domain has always been one of the most time consuming aspects in flow simulation. Grids have to accurately represent the geometry under consideration. Grid points have to be dense enough to resolve the important flow features since the accuracy of a numerical solution depends heavily on the grid spacing. In most cases sufficient knowledge of the flow physics is not known a–priori to increase grid point density in the important regions. Solution based grid adaptation involves distributing the grid points in an optimum fashion to give higher resolution of important flow features. There are three distinct approaches to adapting grids: redistribution, refinement / derefinement, and local increase of the order of the numerical algorithm. Grid redistribution can be done algebraically or by solving partial differential equations. Algebraic methods are fast but can lead to highly skewed grids in complex domains. Algorithms based on elliptic equations on the other hand are much slower but give smooth and close to orthogonal grids. The control functions give a fairly accurate control over grid point distribution. This makes it highly desirable to use elliptic equations to generate solution–adaptive grids.

In the case of a stand–alone grid adaption module, the flow solver is stopped after a few iterations to adapt the grid to the developing solution. A few such iterations result in a completely converged solution. This requires that the grid adaption algorithms be very fast. Further if the flow solver does not account for grid speeds, then the adaption algorithm should interpolate the solution from the earlier grid onto the new grid before the flow solver is restarted. The Parallel Multiblock Adaptive Grid generator (PMAG) presented in this paper, was thus conceived with the motivation of creating a fast grid adaption algorithm that could handle any general multiblock topology.

2. Goals

There should be absolutely no restriction on block connectivity. A block can be be connected to any block including itself, thus supporting a wide range of complex three dimensional topologies.

The algorithm should adapt a multiblock grid concurrently with each block solved in an individual process. These processes could be run on a shared memory parallel machine or distributed over a network of workstations. The algorithm should be scalable. Communication latency should be minimum.

Grid adaption should require minimum user interaction. It should resolve all important flow features. The Neumann boundary conditions should be incorporated to maintain grid quality near body surfaces. Solution interpolation must be included.

3. Grid Adaption

3.1 The elliptic equations

PMAG generates the adapted grids as a solution to the elliptic partial differential equations. Elliptic equations can be written concisely as:

$$\sum_{i=1}^{3}\sum_{j=1}^{3} g^{ij}\,\vec{r}_{\xi^i\xi^j} + \sum_{k=1}^{3} g^{kk}\,P_k\,\vec{r}_{\xi^k} = 0 \qquad (1)$$

The grid point distribution can be controlled by varying the control functions P_k appropriately. It is necessary to maintain the original grid characteristics during grid adaption. Geometric control functions are evaluated from the existing grids using the formulation indicated by Soni [1].

$$P^k_{geom} = \frac{1}{2}\left(\frac{(g_{11})_{\xi^k}}{g_{11}} - \frac{(g_{22})_{\xi^k}}{g_{22}} - \frac{(g_{33})_{\xi^k}}{g_{33}}\right) \qquad (2)$$

3.2 Adaptive weight functions

Adaptive weight functions are computed using the spring analogy form, where the weights are a function of the computational coordinate. The weight function investigated by Thornburg and Soni [3] has proved to detect flow features of varying strengths, including vortices, boundary layers and shocks of disparate strengths and requires almost no user input. This feature is certainly very attractive for a general multiblock grid adaptation algorithm. Relative derivatives are used so that weaker but important features such as vortices are also reflected accurately in the weight functions.

Each conservative flow variable is scaled independently. A small value epsilon is added to all normalizing derivatives, primarily to handle zero velocity regions. This epsilon value also works as a filter for minor disturbances. A boolean sum of the scaled derivatives is used to compute the weight functions in each direction which are then normalized. Weight functions in each direction are then added with another boolean sum to give the final weight function.

$$W = \frac{W^1}{max(W^1, W^2, W^3)} \oplus \frac{W^2}{max(W^1, W^2, W^3)} \oplus \frac{W^3}{max(W^1, W^2, W^3)} \qquad (4)$$

with,

$$W^k = 1 \oplus W^k_\varrho \oplus W^k_{\varrho u} \oplus W^k_{\varrho v} \oplus W^k_{\varrho w} \oplus W^k_e$$

where,

$$W^k_\varrho = \frac{\frac{\left|\varrho_{\xi^k}\right|}{|\varrho|+\epsilon}}{\left(\frac{\left|\varrho_{\xi^k}\right|}{|\varrho|+\epsilon}\right)_{max}} \oplus \frac{\frac{\left|\varrho_{\xi^k\xi^k}\right|}{|\varrho|+\epsilon}}{\left(\frac{\left|\varrho_{\xi^k\xi^k}\right|}{|\varrho|+\epsilon}\right)_{max}}$$

The control functions are computed from the weight function as

$$P^k_{adap} = \frac{(W)_{\xi^k}}{W} \qquad (5)$$

The final control functions are obtained as a weighted sum of the geometric and adaptive control functions.

$$P = C_{geom}P_{geom} + C_{adap}P_{adap} \qquad (6)$$

3.3 Neumann boundary condition

Boundary point movement without loss of geometric details can be accomplished by using NURBS definition of the body surfaces. NURBS definitions of the original body surfaces are created using inverse NURBS formulation. Boundary point movement is achieved by reparameterization of the surface to preserve slope continuity or orthogonality at the boundaries as required.

Boundary layers typically show dense grid packing. It is quite practical to assume that the grid plane next to the boundary would follow the contour of the boundary fairly accurately. This means, orthogonality of grid lines can be easily achieved, if the boundary has a point distribution identical to that on its neighbouring plane. Thus, the algorithm first computes NURBS definition of the grid plane next to the boundary surface. The distribution space of this plane is then used to remesh the boundary NURBS surface thus getting almost identical point distribution on those surfaces. PMAG can also give slope continuity instead of orthogonality if so desired by using a similar strategy [2].

3.4 Solution interpolation

The fast search and interpolation algorithm used in this code takes advantage of the local physical properties by mapping the global geometry in terms of local coordinates α, β, γ of the current cell [5]. The interpolation point x_p is expressed in terms of these local coordinates which fall within a known interval (–1, 1) if x_p lies within the cell. If any of the coordinates fall outside this interval, then the next cell is guessed based on the value of those local coordinates. The next guess cell can be diagonal to the current cell if more than one coordinates falls outside the interval. Further if any of the coordinate is extremely large (say $\alpha >> 2$) then the algorithm can intelligently jump more than one cell in the appropriate direction. This method significantly speeds up the search process.

4. Parallel Implementation

Each block has to be solved in an independent processes. PMAG spawns processes equal to the number of blocks in the topology. Each block is expected to be stored in a separate disk file. This is done to enable concurrent reading and writing of grid files by each process. The user needs to supply the connectivity information for each block. This includes information about shared faces and fixed patches.

Grid lines must be continuous across adjoining blocks. The inter–block faces, edges and vertices which do not describe a definite fixed body must be free to float in the space. This requires that the block faces be solved with an exchange of information across the block faces. Each block has a layer of ghost cells that contain data from the neighboring blocks. This data is updated at intermediate intervals using asynchronous communication.

Each processor computes the control functions and runs the elliptic solver. A global norm is computed after every iteration to determine the convergence criteria. Solution interpolation and boundary point movement using NURBS is done after every *n* iterations as specified by the user.

This algorithm achieves scalability by the use of threads. If excess processors are available, the processes subdivide their domain by unrolling the outmost loop of the solver and the search algorithm. Threads are spawned to work on each subdomain. Use of threads is advantageous in more than one ways. Scalability is the primary advantage. Maximum use of available resources is harnessed by splitting blocks into threads. Controlling number of threads spawned by each process aids in load balancing. The larger blocks can spawn more threads than the smaller blocks. Splitting the domain into smaller chunks leads to better cache performance

Threads are used with MPI although the MPICH implementation is *not* thread safe [7]. Therefore only one thread is active at the time of inter–process communication. The rest of the threads stop during this time. Although, it is not really essential to stop the other threads. A further enhancement to the current version would have other threads continuing the computations with locks on the data while only one thread is dedicated to communications.

4.1 Handling shared faces

To guarantee complete continuity of grid lines across block faces, each block sends a face to its neighbouring block as shown in Figure 1 . The elliptic generator can then run using the face from the neighbouring block as the dirichlet boundary. After every iteration the new updated faces are transmitted to the neighboring block. This way , the face points are solved using the elliptic equations at every step. However, each block solves for the face points separately. Hence it is possible that the face points may not coincide after an iteration.

To get rid of this slight discontinuity, some sequential multiblock codes keep a track of faces, edges and vertices belonging to each block. Connectivity tables are maintained to identify shared vertices edges and faces. At the end of every iteration, these connectivity tables are used to pick out the copies of the common face,edge or vertex and then an averaged (or elliptically solved) value is broadcast to all the owners. Another method es-

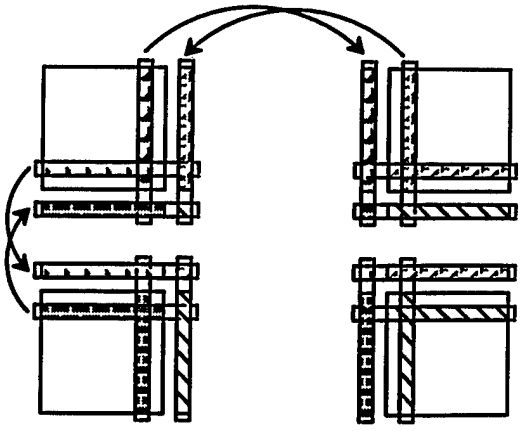


Figure 1 Exchange of information.

pecially amenable in C codes uses pointers to faces, so that only a single copy of a shared face is maintained, with both blocks pointing to the same memory location.

Clearly the later strategy cannot be used when the blocks are being solved on different processors. The first strategy of collecting all copies and then broadcasting an average can work in a parallel implementation but it means a lot of communications. (8 vertices 12 edges and 6 faces per block). Assume a simple structured topology. Each face can be shared by only one other block. One edge is shared by 4 blocks and a vertex is shared by 8 blocks. In such a case, for getting unique face,edge and vertex locations each block needs to communicate with 26 neighbouring blocks even in the simplest topology!

It was argued that since an elliptic system has a 3 point stencil, and since both the blocks have identical copies of all the three faces required for computation of the block boundary face, the face points calculated by the each block using the elliptic system should be the same. Hence, the face points are solved individually using Point Jacobi Iteration (to guarantee a three point stencil). The block interior on the other hand is solved using the tridiagonal system. The control functions for the points on the shared face are also evaluated using only a 3 point stencil guaranteeing that neighboring blocks compute exactly identical locations of the shared points.

Simplicity of the entire scheme is a major advantage. No complicated global connectivity tables need to be maintained. This makes the code extremely flexible, enabling it to handle a wide variety of block topologies, including 0 grids embedded in H grids, Periodic boundary conditions etc.

For the above strategy to work, the basic premise is that all blocks sharing a particular point must have identical copies of its complete stencil. The user input specifies only the blocks that share a face with the current block. This means that a block has no information about its diagonally opposite neighbor. A communication strategy that allows each block to acquire information from its diagonal neighbors is described in the next section.

4.2 Communication of shared faces

Each block knows only the neighboring blocks sharing a face with it. This means the block cannot directly get the corner points from its diagonally opposite block. To overcome this problem, the blocks communicate faces inclusive of the extra points received from the neighboring blocks (shown by the dashed–blocks). This however means that the block/blocks that send the points before receiving the faces from the neighbors would send out void/old points to the neighbors. One way to overcome this problem is to perform communication in a cyclic loop. However this means the entire communication will be sequential and result in large communication latency. The problem can be overcome by simply doing the communication twice. This strategy allows us to use asynchronous communication, so that more than just 2 blocks communicate at any instant. In case of a simple topology discussed in the earlier section each block would now communicate only 12 times. Note that doing the entire communication twice does involve transmission of some redundant information. This could be avoided by sending only the edges and vertices in the second communication. This feature has yet to be implemented in PMAG.

4.3 Parallel multiblock interpolation

As the points are moved the original solution needs to be interpolated onto the new grid. A parallel gird adaptation algorithm supporting general multiblock topologies makes solution interpolation significantly more complex. The grid points can move outside the original domain of a block. In such a case, the block does not have enough information to interpolate the solution and adaptive functions to all its points. Each block now needs to query all other blocks for the points that are no longer within its own domain.

The search algorithm starts with a search and interpolation of all points found within each block. Each process creates a list of points that are not found within its domain. These lists are concatenated into a global list which is broadcast to all processes for search. The processes then search and find the interpolated solutions for the points found within their domains. A final all–reduce over all processes makes the solutions known to all the processes. This operation takes 4*log P communications. A Shift operation (the list of external points are shifted in a circle through all processes) would take P communications to complete. Hence a shift would be faster for number of blocks < 16. Ideally a polyalgorithm should be used to switch methods according to the number of processes. The shift has not yet been implemented in PMAG.

4.4 Results and discussion

PMAG proved very successful in creating adapted grids for the ARL–Missile cases tested as a part of the KTA–12 program [8]. Table 1 shows performance of PMAG for the ARL–Missile case split into 1, 2 or 6 blocks. Table 2 shows the absolute efficiency for 6 blocks run on varying number of processors (compared to runtime for a single block on a single processor). Processes time share when available processors are less than number of blocks. Threads are spawned when excess processors are available (9 and 11). The efficiency drops for the last two cases since NURBS and solution interpolation do not use the threads as yet. All tests were done on a 12 processor SGI Onyx (CPU: 150MHz R4400 with 1.5 GB RAM) . NPARC was used as the flow solver [4].

Table 1 Runtimes for PMAG (mm:ss)

# Blocks	# Procs	Run Time	Speedup
1	1	58:15	1
2	2	32:33	1.8
6	6	13:14	4.4

Table 2 Absolute Speedup and Efficency for 6 blocks

Procs	Run Time	Speedup	Efficiency
1	1:12:20	0.8053	1.2418
2	40:20	1.4442	0.7221
3	26:30	2.1981	0.7326
6	13:14	4.4017	0.7336
9	11:28	5.0799	0.5644
11	10:07	5.7570	0.5227

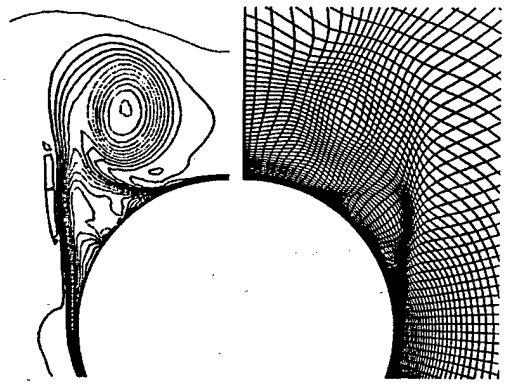


Figure 2 Vortex at x=8.4

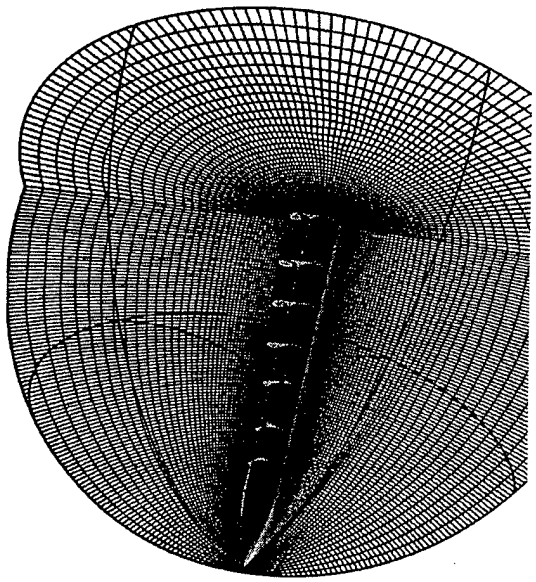


Figure 3 6–Block Grid around the Missile

Figure 4 shows 4 block 2–D geometry of a convergent–divergent nozzle for the Titan rocket. PMAG was used by stacking grids in the third dimension to emulate a 3–D geometry.

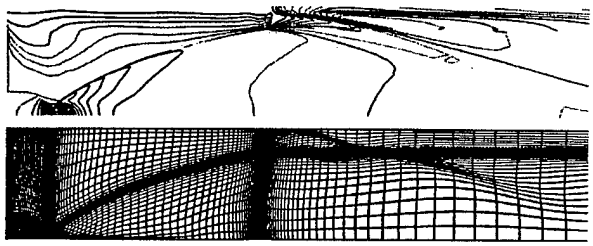


Figure 4 Nozzle solution with adapted grid

Figure 5 demonstrates PMAG's capability to generate smooth elliptic grids around complicated topologies. The grid is a single block O–grid

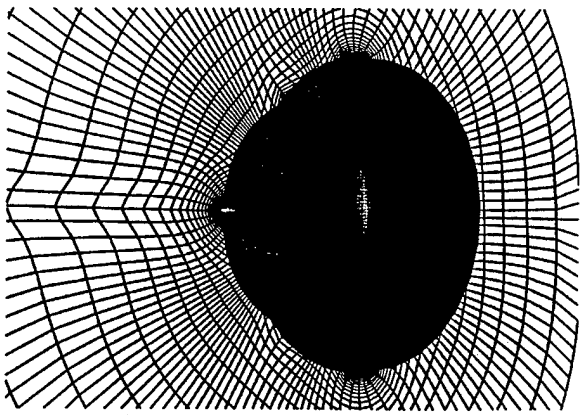


Figure 5 3D geometry with grid

with periodic boundary condition at the kmax–kmin face. The original grid was generated with a

marching algorithm. However that results in highly skewed grids at the corner shown in Figure 6. PMAG was used to smooth the grid. Most elliptic solvers give a crossover at such critical regions and lose the grid point spacing near the boundary.

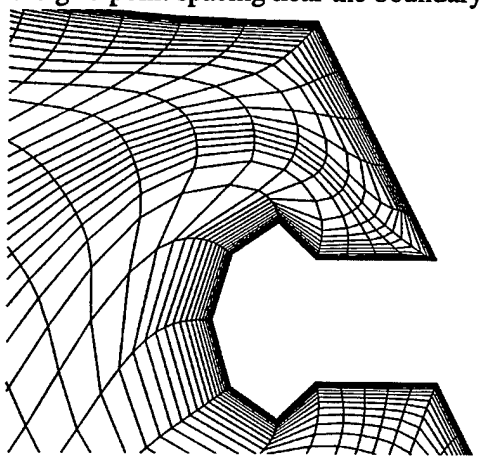


Figure 6 Zoomed view of critical section

PMAG can also be used as a simple elliptic grid generator. Figure 4 shows a complex 5 block grid smoothened using PMAG's elliptic solver. The runtime is compared with a serial multiblock grid generator GUMB which is a derivative of NGP [9]. The speedup shows an absolute efficiency of 0.96 (Table 3).

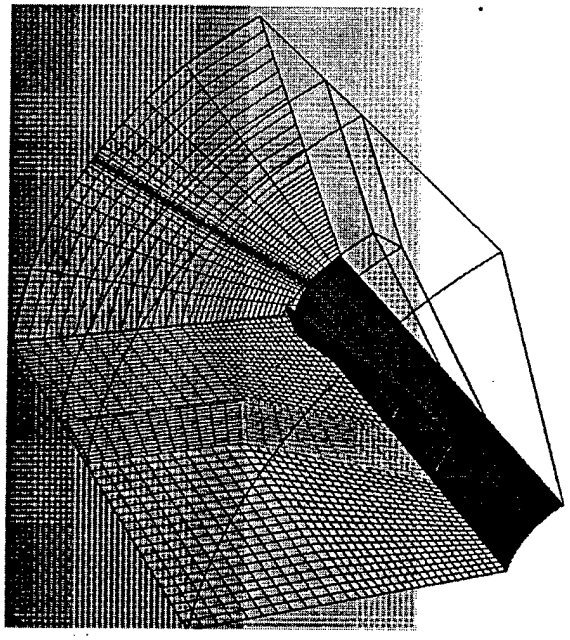


Figure 7 5–Block grid around Missile Fin.

Table 3 Runtime for GUMB v/s PMAG

Algorithm	# procs	Run Time
GUMB	1	2:36:16 hours
PMAG	7	23:55 mins

5. References

[1] Soni, B.K., "Grid Generation: Algebraic and Partial Differential Equations Techniques Revisited", Computational Fluid Dynamics '92, Vol.2, pp. 929–937, Elsevier Science Publishers, 1992.

[2] Apte, M.S., "Parallel Adaptive Grid Generation for Structured Multiblock Domains ", Masters Thesis, Mississippi State University, May 1997.

[3] Thornburgh H.J. and Soni B.K., "Weight Functions in Grid Adaption," *Proceedings of the 4th International Conference in Numerical Grid Generation in Computational Fluid Dynamics and Related Fields.* Swansea, Wales, April 1994.

[4] NASA LeRC and USAF AEDC, NPARC 1.2 User Notes, June 1994.

[5] Stokes, M.L. and Kneile, K.R., "A Three–Dimensional Search/Interpolation Scheme for CFD Analysis",Presented at the *First World Congress on Computational Mechanics*, University of Texas at Austin, September 1986.

[6] Yu, T.Y., "CAGD Techniques in Grid Generation", Ph.D. Dissertation, Mississippi State University,

[7] Gropp, W., Lusk, E. and Skjellum, A., Using MPI: Portable Parallel Programing with the Message–Passing Interface, MIT Press, 1994

[8] Sturek, W.B., Birch, T., Lauzon, M., Clinton, H., Manter, J., Josyula, E. and Soni, B.K., "The Application of CFD to the Predicttion of Missile Body Vortices," AIAA 96–0637, January 1997.

[9] Gaither A., Gaither K., Jean B., Remotique M. et.al., "The National Grid Project: a system overview. ", *Proceedings of the NASA Conference on Surface Modelling, Grid Generation and Related Issues in Computational Fluid Dynamic Solutions,* pp. 423, NASA Conference Publication 3291, May 1995.

5.4 "Parallel Adaptive Grid Generation for Structured multiblock domains", Master's Thesis, Mississippi State University, May 1997.

PARALLEL ADAPTIVE GRID GENERATION FOR STRUCTURED MULTIBLOCK DOMAINS

By

Manoj Shriganesh Apte

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Computational Engineering
in the College of Engineering

Mississippi State, Mississippi

May 1997

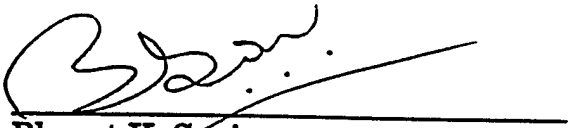
PARALLEL ADAPTIVE GRID GENERATION FOR

STRUCTURED MULTIBLOCK DOMAINS

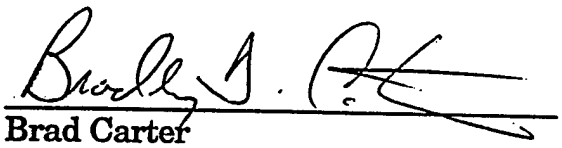
By

Manoj Shriganesh Apte

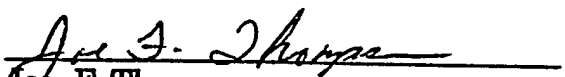
Approved:



Bharat K. Soni
Professor of Aerospace Engineering
(Major Professor and Thesis Director)



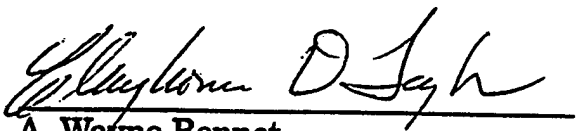
Brad Carter
Professor of Computer Science and Engineering
(Graduate Coordinator of Computational Engineering)



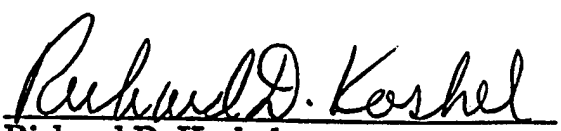
Joe. F. Thompson
Professor of Aerospace Engineering

Anthony Skjellum
Associate Professor of Computer Science

Boyd Gatlin
Associate Professor of Aerospace Engineering



A. Wayne Bennet
Dean of the College of Engineering



Richard D. Koshel
Dean of the Graduate School

Name: Manoj Shriganesh Apte

Date of Degree: May 10, 1997

Institution: Mississippi State University

Major Field: Computational Engineering

Major Professor: Dr. Bharat K. Soni

Title of Study: PARALLEL ADAPTIVE GRID GENERATION FOR STRUCTURED MULTIBLOCK DOMAINS

Pages in Study: 79

Candidate for Degree of Master of Science

The objective of this study was to develop a parallel structured grid adaptation algorithm for general three dimensional multiblock domains. Grid blocks are computed in individual processes that maybe distributed over multiple processors. MPI is used for message passing. The grids are redistributed as a solution of the elliptic partial differential equations. Weight function is computed as a boolean sum of the scaled gradients and curvatures of all flow variables. Neumann boundary condition has been implemented using NURBS to maintain geometric fidelity. Surface definitions are generated by using inverse NURBS formulation. A parallel multiblock solution interpolation algorithm has been incorporated to guarantee accurate adaptation. The algorithm can also be used simply as a multiblock elliptic grid generator.

Several test cases with a variety of flow characteristics are presented to demonstrate the capabilities of the algorithm. Comparisons made with existing algorithms show distinct advantages gained by using this approach.

DEDICATION

To my parents

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to my major advisor Dr. Bharat K. Soni for his kind guidance and support. His constant encouragement and appreciation of individual effort has been a major driving force throught this project. I would also like to thank my committee members Dr. Anthony Skjellum and Dr. Joe Thompson for their invaluable advice and guidance.

I would like to thank my friends and colleagues at the Engineering Research Center, whose support has been an invaluable asset. A special thanks to Dr. Roy Koomullil, Dr. Hugh Thornburg, Dr. Tzu Yi Yu and Puri Bangalore for their help and guidance.

I am grateful to the Engineering Research Center (ERC) and the Army Research Labs (ARL) for funding this project.

The greatest acknowledgement goes to my parents and my family for their constant encouragement and understanding through all these years of my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

Background

The past decade has seen a phenominal increase in the thrust towards numerical simulation of complex three dimensional flows associated with realistic configurations. Computational flow simulation involves numerical solution of the equations representing the actual physics. The numerical methods for solving various flow equations have been available for a long time. However, the capability to handle complex realistic configurations has been paced mainly by the ability to numerically discretize the domain of interest into a mesh of points.

Grid generation is the starting point of any computational fluid dynamics simulation. Creation of a suitable grid over a complex multiblock domain has always been one of the most time consuming aspects in flow simulation. Grids have to accurately represent the geometry under consideration. Grid points have to be dense enough to resolve the important flow features since the accuracy of a numerical algorithm depends heavily on the grid spacing (a sparse grid in a region of high flow gradients results in a loss of important flow features). In most cases sufficient knowledge of the flow physics is not known a priori to increase grid point density in the important regions. Using a uniformly dense mesh would result in excessive usage of computer memory and CPU time. Further, such high resolution is required in a very small area of

the flow region. In most cases the initial grid can resolve only the general features of the solution.

Grid adaptation involves distributing the grid points in an optimum fashion to give higher resolution of important flow features. There are three distinct approaches to adapting grids: redistribution, refinement / derefinement, and local increase of the order of the numerical algorithm. Although the last method requires no change in the grids, it requires highly complex solution algorithms. Mesh refinement involves adding points in the region of high gradients [28][29]. This method guarantees that the global error will not increase since no depletion of points occurs in other regions. However, it requires maintainace of complex data structures to allow addition of points anywhere in the region. If starting from a uniform grid, Adaptive Mesh Refinement (AMR) results in cells that are more or less isometric (Aspect Ratio $\simeq 1$). This constraint is particularly unattractive in case of boundary layer cells requiring high aspect ratio. The true advantage of using AMR methods lies in topologies involving largely varying length scales. Astrophysics, Ocean modelling etc. would be typical examples of such topologies.

Redistribution is achieved by moving grid points locally or globally to regions of higher gradients. This method does not increase memory or CPU requirement since total number of grid points remains the same. Boundary layers can be resolved very well with optimum number of points. Redistribution works the best for topolgies that do not contain length scales that differ by many orders of magnitude. Most aerospace geometries have length scales that do not need an AMR approach to get the best efficiency. Simplicity of solution algorithms and data structures required is a major advantage of this method.

Grid redistribution for structured grids can be done either algebraically or by solving partial differential equations. The algebraic methods are very fast but can result in highly skewed grids in complex geometries. Elliptic partial differential equations on the other hand are much slower but give smooth and close to orthogonal grids even after adaptation. This makes these methods highly desirable.

Overview of Existing work

Several grid adaptation algorithms for structured grids are available at present. These algorithms can be broadly classified into two types: those based on algebraic methods, and those based on solution of PDEs. SAGE [9] and Yang's algorithm [16] are examples of the algebraic grid adaptation methods, while the EAGLE code [5] and Thornburg's work [2] are based on solution of elliptic partial differential equations.

SAGE (Self–Adaptive Grid Code) is based on the algebraic method outlined by Nakahashi and Deiwert [8]. The adaptive grid procedure is based on grid–point redistribution through local error minimization using the spring analogy form. Davis and Venkatapathi [9] evolved this method into a flexible tool for adapting two–dimensional and three–dimensional grids with multiple blocks. Multidimensional adaptation is split into a series of one dimensional adaptations along each coordinate line. adaptation is done line by line and plane by plane for all planes specified. A weight function proportional to the derivative of a flow property is used. Torsional force term is used to achieve orthogonality and smoothness of grid. The force is calculated as torsion between adjacent grid lines. Although this method is fast, the grids generated by this method are not always smooth. In case of multiblock grids, the code transfers the shared face from first block to second. The common face is evaluated

only in the first block. This method would achieve C^0 continuity but cannot guarantee slope continuity. Further, a lot of user input is required for using the code which makes it difficult to use with large number of blocks.

Yang [16] has created a general purpose adaptive grid generation system based on algebraic redistribution of points using NURBS. A weight function evaluated with properly weighted boolean sum of various flow filed characteristics is used. It applies inverse NURBS formulation to compute NURBS control points for the surfaces. This maintains geometric fidelity of the adapted grids. The multidimensional grid adaptation is split into a series of two–dimensional grid adaptations along one of the computational coordinate lines. The two–dimensional algebraic grids are smoothened by a few iterations of an elliptic solver. This dilutes the adaptive effect and does not guarantee smoothness of grid lines along the third dimension. Yang's algorithm has produced excellent results on two dimensional grids and simple three dimensional regions. There is no capability to adapt multi–block grids.

The EAGLE code is a composite grid generator supporting completely general configurations [5]. This code uses elliptic grid generation system with automatic evaluation of control functions, either directly from the boundary or from the boundary point distributions. Neumann boundary conditions have been provided in addition to control functions to achieve orthogonality at the boundaries. Boundary surfaces are represented as splines. The new boundary points are located by orthogonal projection of adjacent points using Newtons iteration. Slope continuity across blocks is guaranteed by forcing the grid lines on both sides to be orthogonal to the shared face. The EAGLE adaptation routine does not have solution interpolation built in. Hence grid adaptation must

be done either at every time step or the PDE solver must account for grid speeds [6][7].

Thornburg's approach [2] also involves grid adaptation using solution of elliptic differential equations. A newly developed weight function employing Boolean sums is utilized to represent the local truncation error. A NURBS representation is employed to define block surfaces for bounday point redistribution. A coupled strongly implicit procedure (CSIP) as described by Ghia et.al. [4] has been implemented for the solution of discretized equations. Upwind differencing, with biasing based on the sign of the forcing functions, as well as central differencing has been implemented for the derivative terms. The first order upwind differencing increases the stability of the procedure. A hybrid upwind/central differencing scheme has been implemented. The interpolation procedure employed was the one used for the non–matching block to block interface capability of the NPARC code [18]. The code shows excellent results on resolving various flow features. However, Neumann boundary conditions can be applied only if the user supplies the NURBS definitions of the original geometry. It is extremely slow and there is no multiblock handling capability. The current work is primarily based on the weights developed by Thornburgh and Soni. More detailed description of these is included in Chapter II.

Motivation

Most realistic configurations consist of complex multiblock topologies. The adaptation algorithm thus requires to have the capacity to handle topologies with any kind of block connectivities. Grid lines should be completely continuous across boundaries without forcing orthogonality to achieve slope continuity.

Typically when using grid adaptation, the solver has to be stopped after a certain number of iterations for modifying the grids using a grid adaptation algorithm. This cycle is repeated a few times until complete convergence is achieved.

Experience with a realistic missile geometry (> 1 million points) showed that adaptation based on elliptic grid generation yields much better results for a flow containing vortices as well as shocks. However, the CSIP grid adaptation algorithm took about a third of the total time required for computing the final solution (order of 10 hours for each adaptation cycle on a million point grid). This being quite a significant amount of time, attention was focused towards creation of faster algorithms for grid adaptation.

Adapted grids get skewed near body surface if the boundary points are not moved during redistribution. Dirichlet boundary conditions keep the boundary points fixed. This makes it necessary to design an algorithm that allows Neumann boundary conditions.

Intermediate solution interpolation is essential to guarantee that grid points are concentrated in the correct regions. Parallel multiblock grid adaptation further complicates the solution interpolation procedure. Moving block faces require each block to query other blocks for points that have moved outside its domain. A fast parallel multiblock interpolation routine is therefore essential in the algorithm.

Solution speed can be dramatically increased with parallel processing. The growing abundance of relatively cheap high performance parallel machines makes it more practical to design algorithms that aim at solving voluminous problems on multiple processors. A parallel grid adaptation code was hence conceived.

Overview of the Present Work

The Parallel Multiblock Adaptive Grid generation (PMAG) algorithm has been designed to handle a general multiblock topology. There is absolutely no restriction on block connectivity. A block can be be connected to any block including itself, thus supporting a wide range of complex three dimensional topologies.

The algorithm is designed to adapt a multiblock grid concurrently with each block solved in an individual process. These processes can be run on a single parallel machine or distributed over a network of workstations.

MPI is used as the message passing interface. The advantages of using MPI are that it has the capability to have asynchronous communications to hide communication latency, it is supposed to work over a heterogenous network, and it is now a standard.[26][27]

Grid adaptation in PMAG is based on the boolean sum of weight functions suggested by Thornburg, et.al. [2]. These weight functions have demostrated the capacity to detect shocks of differing strengths, primary and secondary vortices, and shear layers adequately. Enhancements have been made for evaluating globally normalized weights for multiblock grids as suggested by Thornburg.

A simple tridiagonal solver is used for generating the elliptic grids. The solver is much faster than CSIP, and has shown no major disadavantage in the quality of grids achieved.

Neumann boundary conditions have been incorporated to maintain grid quality near body surfaces. PMAG is designed to allow boundary point movement by defining the boundaries as NURBS surfaces. This guaratees that the geometric definition is preserved accurately.

Solution interpolation is based on the method suggested by Stokes [20]. Several enhancements have been made for faster and more robust interpolation. The same algorithm has been extended for parallel multiblock solution interpolation.

PMAG has been designed primarily on the SGI Power Challenge platforms. It has the capacity to do concurrent computing at two levels. Each MPI–process can further spawn threads on a parallel shared memory machine giving finer grained parallelism. The ability to spawn threads helps in many ways. Better scalability, a simplistic load balancing technique, better cache utilization by reducing array sizes being the most significant of the advantages. The code has also been ported to Sun workstations (without threads).

Organization

The subsequent chapters take a detailed look at each of the aspects mentioned here.

Chapter II discusses the theoritical development of the basic adaptation algorithm based on solution of elliptic partial differential equations. This includes discussion of the geometric and solution adaptive controls, implementation of NURBS and a fast solution interpolation algorithm. Most of these topics are based on earlier work by others, with a few modifications to improve performance.

Chapter III discusses the major issues involved in implementing the code in a parallel paradigm. Multiblock issues such as strategies for shared face evaluation, inter–block communication, and solution interpolation are discussed in detail. Implementation of multi–threading for shared memory machines (SGI Power Challenge series) is also discussed. All the strategies

and algorithms dicussed in this chapter are original contributions of the author.

Chapter IV includes the results and discussion on various test cases. Conclusions and areas of possible future work have also been suggested.

A users guide has been provided for reference in Appendix B.

CHAPTER II

THEORITICAL DEVELOPMENT FOR GRID ADAPTATION

Introduction

Traditionally, smooth structured grids have been created as solutions of a set of partial differential equations. The PDEs can be either parabolic, hyperbolic or elliptic. Elliptic equations are the most stable and have a smoothing effect on the grids. In most cases this results in smooth, near orthogonal grids even if the boundary has sharp corners. This is desirable for most flow solvers. The control functions in the Poisson equations make it possible to cluster the grids in region of interest, and thus provide an elegant way to redistribute grids to resolve important flow features.

The Elliptic System of Equations

A grid is smooth if it is second order continuous. Mathematically if ξ, η, ς are curvilinear coordinates along the grid lines, then the solution to the Poisson equations

$$\begin{aligned} \nabla^2 \xi &= \phi \\ \nabla^2 \eta &= \theta \\ \nabla^2 \zeta &= \delta \end{aligned} \qquad (2.1)$$

should give smooth grids as long as the functions ϕ, θ and δ are smooth and continuous. Structured elliptic grid generation involves generation of grids that satisfy the above equations. To be usable for generating cartesian coordi–

nates X, Y and Z, we must first cast the equations to have X, Y and Z as the dependent variables. This leads to a system of equation of the type,

$$\sum_{i=1}^{3}\sum_{j=1}^{3} g^{ij}\,\vec{r}_{\xi^i\xi^j} + \sum_{k=1}^{3} g^{kk}\,P_k\,\vec{r}_{\xi^k} = 0 \qquad (2.2)$$

Where:

r_{ij} : Position vector,

g_{ij} : Contravariant metric tensor,

ξ^i : Curvilinear coordinate, and

P_k : Control Function.

Once cast in this form, the differential equations are discretized using central difference form. Discretized equations give a system of linear equations that can be cast in a tridiagonal form. Solution is obtained by iteratively solving the tridiagonal system of equations. The details of this method are discussed in Appendix A.

Geometric Control Functions

Laplace equations, with the forcing functions identically equal to zero tend to create equi–distributed grids. However in case of viscous flows closely packed grid lines are critical for resolving the boundary layer. Further, Laplace grids also move grid lines away from the concave boundaries, and towards the convex boundaries. This behaviour is highly undesirable and hence forcing functions are used that preserve grid line spacing and orthogonality near the body surface.

Grid line spacing would be maintained if forcing functions were derived by solving the equation (2.2) for P_k. Various methods of evaluating control functions have been investigated [11]. All these methods assume orthogonality

of grid lines as a desirable property. The method discussed by Soni [1] is fast and easy to implement.

Consider the three dimensional elliptic grid generation system :

$$\sum_{i=1}^{3}\sum_{j=1}^{3} g^{ij}\,\vec{r}_{\xi^i\xi^j} + \sum_{k=1}^{3} g^{kk}\,P_k\,\vec{r}_{\xi^k} = 0 \qquad (2.3)$$

In order to evaluate the forcing functions P_k, k=1,2,3 the usual practice is to take a dot product of the equation (2.3) with $r_\xi{}^q$, q=1,2,3 and write the equation as :

$$\sum_{i=1}^{3}\sum_{j=1}^{3} g^{ij}\,\vec{r}_{\xi^i\xi^j}\cdot\vec{r}_{\xi^q} + \sum_{k=1}^{3} g^{kk}\,P_k\,\vec{r}_{\xi^k}\cdot\vec{r}_{\xi^q} = 0 \qquad (2.4)$$

Using the definitions from the earlier section, we can rewrite the equation using only the metric terms and their derivatives,

$$\sum_{i=1}^{3}\sum_{j=1}^{3} g^{ij}\,(g_{iq})_{\xi^j} + \sum_{k=1}^{3} g^{kk}\,P_k\,g_{kq} - \sum_{i=1}^{3}\sum_{j=1}^{3} g^{ij}\,(\vec{r}_{\xi^i}\cdot\vec{r}_{\xi^j\xi^k}) = 0 \qquad (2.5)$$

If grid lines are assumed to be orthogonal, $g_{ij} = 0$ if ($i \neq j$). Equation (2.5), can then be simplified as :

$$\sum_{i=1}^{3} g^{ii}(\,\vec{r}_{\xi^i\xi^i} + \delta_{iq}P_q\cdot\vec{r}_{\xi^q}\,) = 0 \qquad q=1,2,3 \qquad (2.6)$$

$$\therefore \quad P_k = \frac{1}{2}\frac{d}{d\xi^k}\left(\ln\frac{g_{kk}}{g_{ii}\,g_{jj}}\right) \quad , (i,j,k)cyclic \qquad (2.7)$$

The same formula can also be expressed as shown in equation (2.8)

$$\phi = \frac{1}{2}\left(\frac{(g_{11})_\xi}{g_{11}} - \frac{(g_{22})_\xi}{g_{22}} - \frac{(g_{33})_\xi}{g_{33}}\right)$$

$$\theta = \frac{1}{2}\left(\frac{(g_{22})_\xi}{g_{22}} - \frac{(g_{33})_\xi}{g_{33}} - \frac{(g_{11})_\xi}{g_{11}}\right)$$

$$\psi = \frac{1}{2}\left(\frac{(g_{33})_\xi}{g_{33}} - \frac{(g_{11})_\xi}{g_{11}} - \frac{(g_{22})_\xi}{g_{22}}\right) \qquad (2.8)$$

This is the form used in the algorithm for computing the geometric weight functions. The metrics g_{11}, g_{22}, g_{33} are already available, and hence the evaluation of forcing functions is very easy.

Solution Based Adaptive Control Functions

Solution based grid adaptation involves clustering grid points in the regions of high flow gradients to achieve better resolution of flow features, and reduce the truncation errors.

Grids can be viewed as a collection of points interliked by springs. The elliptic equations are obtained by minizing energy for such a system using the variational principle. If all spring constants are identical, the resulting PDE is the Laplace equation and the solution is a equidistributed grid. On the other hand, desired grid point distributions can be obtained by varying the spring constants appropriately.

The basic idea of adaptive redistribution originates from the principle of equidistribution of error [15]. The point spacing must be inversely proportional to some measure of the error w in the solution.

$$x_\xi \; w(\;) = const \qquad (2.9)$$

The weight function w can either be a function of the computational coordinate ξ (Spring Analogy form) or of the actual physical location x (Point Den-

sity form). Since the solution is a function of the physical coordinates, it would be preferable if the weight function were derived as a function of the same. However, since solution data is available only at the grid points, it is much easier to find the weight function at grid points. Hence, the spring analogy approach is followed. However, it adds an overhead of having to interpolate/ recalculate the weight functions after every few iterations as the points move through physical space.

The formulation of such weight functions that accurately reflect the truncation error in a region is thus the most critical part in grid adaptation. Derivatives of pressure or density are widely used in contruction of weight functions. For a given physical property u the weights are typically calculated using a formula of type.[13][14][15][17]

$$w(\xi) = 1 + \alpha \, |u_x(\xi)| + \beta \, |u_{xx}(\xi)| \qquad (2.10)$$

Although such weights have proved very successful in resolving inviscid flow features they are less capable in resolving viscous features such as boundary layer and vortices. Further, such functions require the user to input relative weights α,β. For a multiblock grid adaptation algorithm aimed at complex geometries, it is important to have a weight function that needs minimum user input.

The weight function investigated by Thornburg and Soni [2][3] has proved to detect flow features of varying strengths, including vortices, boundary layers and shocks of disparate strengths and requires almost no user input. This feature is certainly very attractive for a general multiblock grid adaptation algorithm. Relative derivatives are used so that weaker but important features such as vortices are also reflected accurately in the weight functions.

Each conservative flow variable is scaled independently. A small value epsilon is added to all normalizing derivatives, primarily to handle zero velocity regions. However, this epsilon value also works as a filter as described later. A boolean sum of the scaled derivatives is used to compute the weight functions in each direction which are then normalized. Weight functions in each direction are then added with another boolean sum to give the final weight function.

$$W = \frac{W^1}{max(W^1, W^2, W^3)} \oplus \frac{W^2}{max(W^1, W^2, W^3)} \oplus \frac{W^3}{max(W^1, W^2, W^3)} \qquad (2.11)$$

where,

$$W^k = 1 + \frac{\frac{|\varrho_{\xi^k}|}{|\varrho|+\varepsilon}}{\left(\frac{|\varrho_{\xi^k}|}{|\varrho|+\varepsilon}\right)_{max}} \oplus \frac{\frac{|(\varrho u)_{\xi^k}|}{|(\varrho u)|+\varepsilon}}{\left(\frac{|(\varrho u)_{\xi^k}|}{|(\varrho u)|+\varepsilon}\right)_{max}} \oplus \frac{\frac{|(\varrho v)_{\xi^k}|}{|(\varrho v)|+\varepsilon}}{\left(\frac{|(\varrho v)_{\xi^k}|}{|(\varrho v)|+\varepsilon}\right)_{max}} \oplus \frac{\frac{|(\varrho w)_{\xi^k}|}{|(\varrho w)|+\varepsilon}}{\left(\frac{|(\varrho w)_{\xi^k}|}{|(\varrho w)|+\varepsilon}\right)_{max}}$$

$$\oplus \frac{\frac{|\varrho_{\xi^k\xi^k}|}{|\varrho|+\varepsilon}}{\left(\frac{|\varrho_{\xi^k\xi^k}|}{|\varrho|+\varepsilon}\right)_{max}} \oplus \frac{\frac{|(\varrho u)_{\xi^k\xi^k}|}{|(\varrho u)|+\varepsilon}}{\left(\frac{|(\varrho u)_{\xi^k\xi^k}|}{|(\varrho u)|+\varepsilon}\right)_{max}} \oplus \frac{\frac{|(\varrho v)_{\xi^k\xi^k}|}{|(\varrho v)|+\varepsilon}}{\left(\frac{|(\varrho v)_{\xi^k\xi^k}|}{|(\varrho v)|+\varepsilon}\right)_{max}} \oplus \frac{\frac{|(\varrho w)_{\xi^k\xi^k}|}{|(\varrho w)|+\varepsilon}}{\left(\frac{|(\varrho w)_{\xi^k\xi^k}|}{|(\varrho w)|+\varepsilon}\right)_{max}}$$

Addition of the weights in different directions with a boolean sum as shown in Equation (2.11) is an important unique feature of this method. Figure 2.1 shows a comparison of grid adaptation with and without a boolean sum of the three weights. The boolean sum makes the grid adaptation uniform in all directions, making it possible to resolve circular vortex structures.

The control functions are derived from the weight function using the following formula:

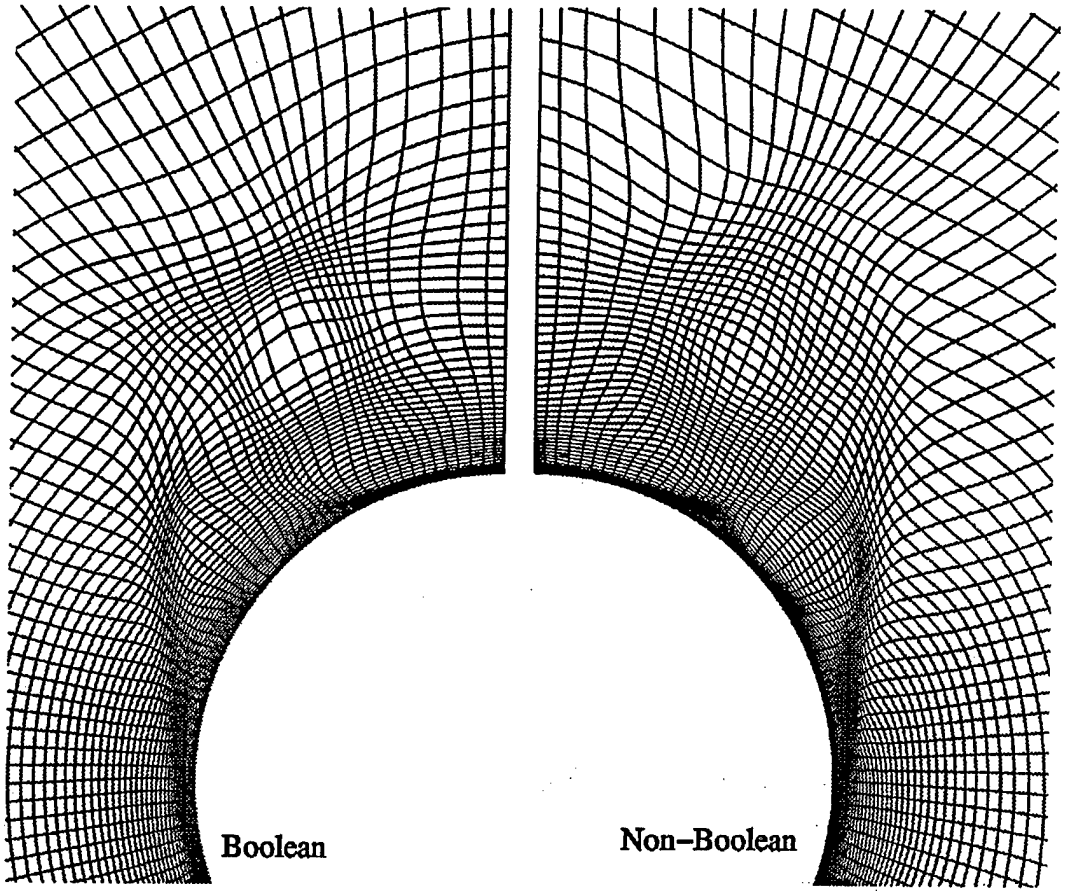


Figure 2.1 Comparison of weights

$$P_i = \frac{(W_i)_{\xi'}}{W_i} \tag{2.12}$$

In order to conserve the geometrical characteristics of the original grid, the control functions used for the elliptic solver are formulated as a combination of the geometric control functions and the adaptive control functions [15].

$$P = C_{geom}P_{geom} + C_{adap}P_{adap} \tag{2.13}$$

Figure 2.2 shows the weight function detecting a vortex behind a missile at 14 degree angle of attack.

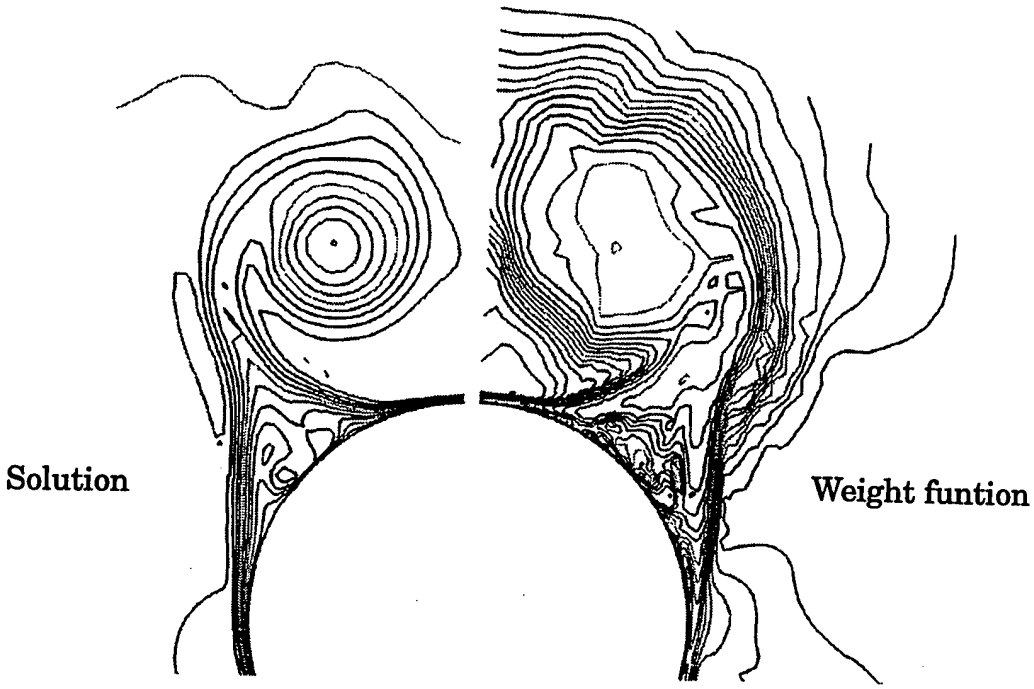


Figure 2.2 Solution based weight function

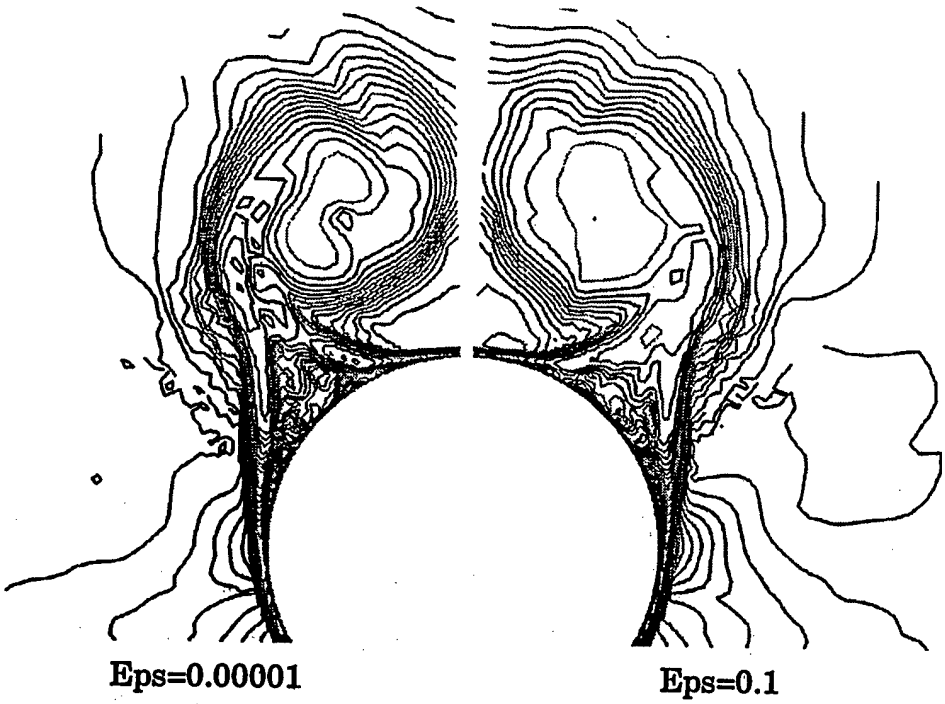


Figure 2.3 Effect of Epsilon

The small value epsilon used in the normalizing derivatives also helps in filtering out small disturbances in the flow field that cause the grid to be concentrated in the non–critical regions. Increasing the value of epsilon reduces the sensitivity of the weight function to small disturbances in the flow field. Figure 2.3 illustrates the effect of increasing the value of epsilon on the weight function.

Boundary point movement using NURBS patches

Strong grid adaptation near body surfaces leads to highly skewed grids if the boundary points are not moved. The most important concern in moving boundary points is preserving geometric fidelity. As shown in Figure 2.4 simple linear interpolation results in the loss of geometry information that is highly undesirable.

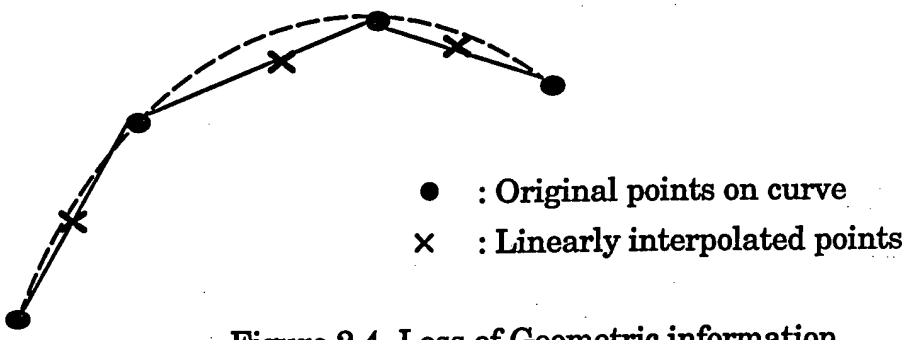


Figure 2.4 Loss of Geometric information

Boundary point movement without loss of geometric details can be accomplished by using NURBS definition of the body surfaces. The original body surfaces are used to create NURBS definitions using inverse NURBS formulation. Boundary point movement is achieved by reparameterization of the surface to preserve slope continuity or orthogonality at the boundaries as required. In this section the details of the implementation of Neumann

boundary condition will be discussed. A quick overview about NURBS and a discussion of the terminology is presented in the next sub–section.

Background on NURBS

When creating grids for real world configurations, the curves and surfaces of the body are given either in discretized form or as parametrized representations using Bezier surfaces, B–Splines or NURBS. Non–Uniform Rational B–Splines referred to as NURBS, have become standard tools for representing and designing free–form as well as standard analytic shapes of Computer Aided Geometric Design (CAGD). A few reasons for popularity and widespread use of NURBS are that they are true generalizations of nonrational B–Spline forms as well as rational and nonrational Bezier curves and surfaces. They offer a common mathematical form for representing both analytical shapes as well as freeform shapes. Evaluation of NURBS is reasonably fast, and computationally stable. NURBS are invariant under scaling, rotation, translation and sheer as well as parallel and perspective projection. [22][23]

NURBS give a parametric definition of any geometric entity in terms of control polygons, knot vectors and associated weights. The order of a NURBS dictates the order of the B–Spline curves used for representing the geometric entity.[24]

Analytically , NURBS surface is defined by the equation (2.14), where, d_{ij} is the control mesh for the surface, w_{ij} are the weights associated with each point on the control mesh, $\{ u_o \ldots u_{m+k} \}$ and $\{ v_o \ldots v_{n+l} \}$ are the knot vectors along each coordinate direction and $N_i^k(u)$ are the normalized B–spline basis functionals.

$$s(u,v) = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} w_{ij} d_{ij} N_i^k(u) N_j^l(v)}{\sum_{i=0}^{m}\sum_{j=0}^{n} w_{ij} N_i^k(u) N_j^l(v)} \qquad \begin{matrix} u \in [u_{k-1}, u_{m+1}] \\ v \in [v_{l-1}, v_{n+1}] \end{matrix} \qquad (2.14)$$

The normalized B–spline basis functionals are evaluated as,

$$N_i^k(u) = \frac{u - u_i}{u_{i+k-1} - u_i} N_i^{k-1}(u) + \frac{u_{i+k} - u}{u_{i+k} - u_{i+1}} N_{i+1}^{k-1}(u)$$

$$N_i^1 = \begin{Bmatrix} 1, \text{if } u_i \le u \le u_{i+1} \\ 0, otherwise \end{Bmatrix} \qquad (2.15)$$

The De Boor algorithm is used to evaluate the physical coordinates of the surface.[22]

Implementation of Neumann Boundary Condition

Neumann boundary condition can be implemented by allowing the boundary points to move such that the slope at the boundary is maintained. For resolving viscous layers, orthogonality of grid lines at the body surface is a highly desirable feature.

Boundary layers typically show dense grid packing. It is quite practical to assume that the grid plane next to the boundary would follow the contour of the boundary fairly accurately. This means, orthogonality of grid lines can be easily achieved, if the boundary has a point distribution identical to that on its neighbouring plane. Thus, the algorithm designed to achieve orthogonality moves points on the boundary such that the point distribution coincides to its immediate neighbour.

Grid points define all the surfaces with discrete data. The projection and inversion algorithm created by Yu [24] creates NURBS definitions of these

surfaces. This routine returns a set of control points d_{ij}, weight functions w_{ij} , and the knot vectors $\{ u_o \ldots u_{m+k} \}$ and $\{ v_o \ldots v_{n+l} \}$. These parameters are then used to remesh NURBS surfaces. to achieve orthogonality at the boundaries.

For preserving geometric definition of the body, grid points are moved only by moving them in the distribution space. The new distribution mesh is then remapped into the physical space using the NURBS definitions created earlier. For orthogonality, the distribution mesh of the boundary surface must be exactly identical to that of its neighbour. The distribution space of NURBS is not related to the arclength distribution space. It depends on the position of knot vectors, control polygons and the weights (Figure 2.5). The (s,t) values

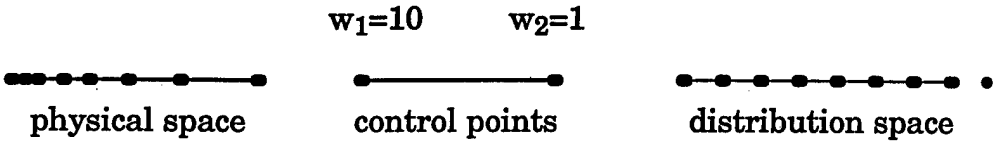


Figure 2.5 Distribution Space for NURBS

for each point on the surface have to be calculated from the NURBS definition using Newtons method [24]. This requires us to create a NURBS definition of the surface neighbouring the boundary. However we already assumed that the neighbouring plane must have a geometry almost identical to that of the boundary. The principle of invariance of NURBS dictates that if one surface can be mapped into another by a simple linear transformation, then they have the same weights and knot vectors. Hence, we only need a new set of control points to be computed for the face next to the boundary. Using that and the weights and knots of the boundary face, an algorithm computes the (s,t) distribution for the neigbouring surface. This distribution is used to remesh the boundary surface with its own set of control points.

If instead of orthogonality, only slope continuity is needed at the NURBS boundary, distribution spaces of two neighbouring planes are calculated instead of one, as described above. The distribution space for the boundary is then calculated by extrapolating the slopes from the earlier planes.

The surface remeshing subroutine can be called after every few iterations to adjust boundary points.

Search and Interpolation

The weight function controlling grid point movement is computed from the solution. Since we follow the spring analogy form, the weight distribution gets distorted as the points move. It is therefore neccessary to interpolate the solution and weights from the original mesh onto the new mesh after every few iterations. This requires a good search and interpolation algorithm integrated with the adaptive algorithm to guarantee accurate clustering. Figure 2.6 below compares grid adaptation with and without interpolation. Without interpolation, the grids tend to keep moving towards the center. Also, the clustering near the feeder of the vortex moves away from the actual region of interest.

Solution interpolation is also important to transfer the solution obtained on an unadapted mesh to the new adapted mesh. Solution interpolation can be very time consuming, and hence a fast method for accurate solution interpolation is required for best results.

Several search and interpolation algorithms have been investigated in the literature [19]. A standard search algorithm to get the location of random points on a grid, involves sequential or exaustive search. This search moves systematically through the entire grid, scanning each cell until it finds the correct cell containing the point. However, a sequential search is very inefficient

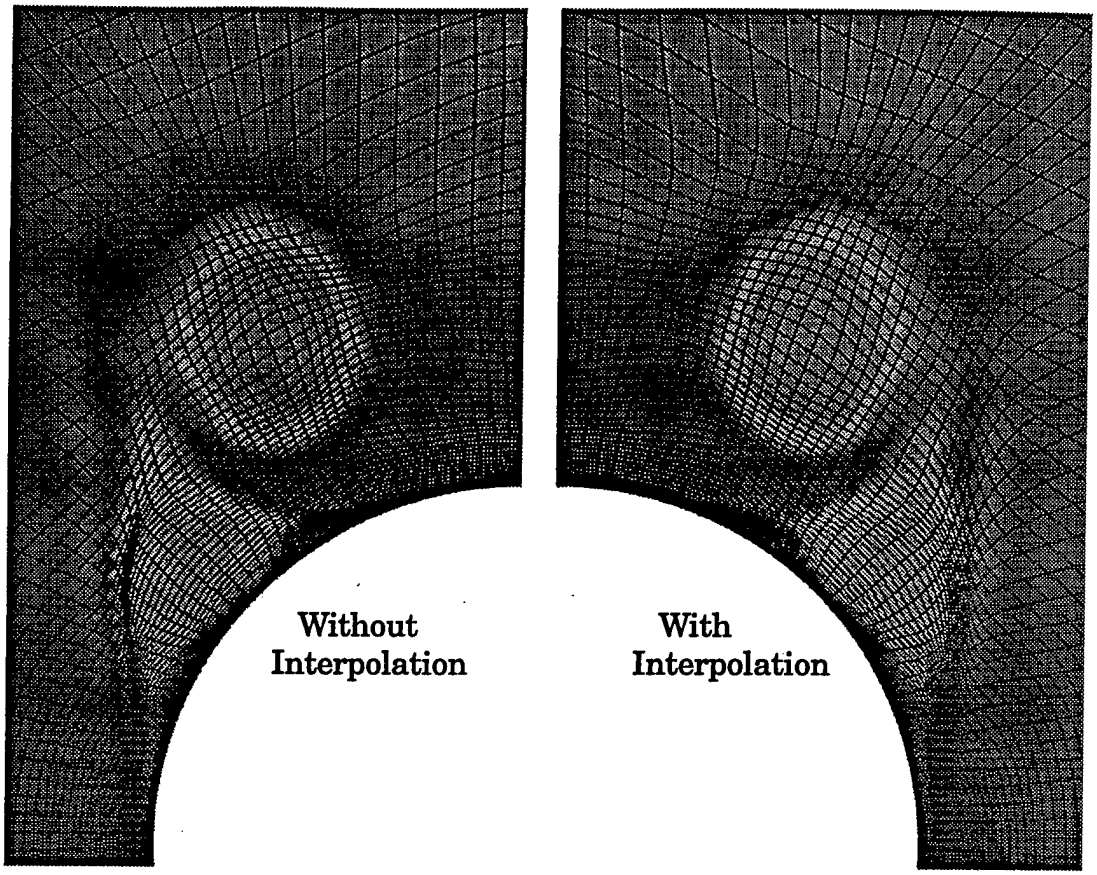


Figure 2.6 Effect of Solution Interpolation

and expensive since it does not use any grid structure information. A search of this kind takes more time than the tridiagonal solution.

Two fast interpolation schemes have been investigated by Mastin [19]. The first algorithm is based on distance calculation. Suppose a point Q has to be searched on a grid starting at vertex P of a cell. The algorithm computes the distance of Q from the neighbours closest to P, and moves to the neighbour that is closer to Q than P. This step is repeated untill a vertex R is found that is closer to the Q than any other vertex. This is a simple algorithm, which is definitely faster than the sequential search. However, the paper notes that this algorithm does not guarantee finding the closest point on highly skewed grids. The algorithm also does not have the ability to step diagonally.

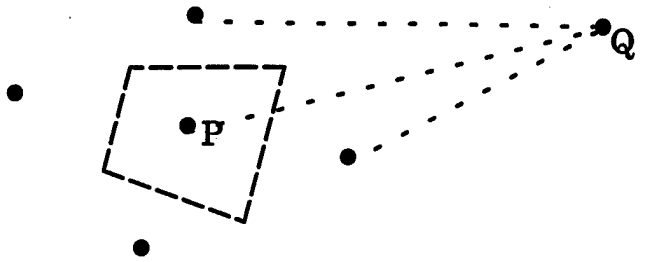


Figure 2.7 Mastin's Interpolation Scheme I

In case of skewed grids, Mastin suggests an alternative method in which, given a starting cell C, the algorithm determines the cell edge such that the line L along that edge separates the point Q from the rest of the cell. The cell D on the other side of L is then taken as the next approximation and the algorithm continues untill Q is inside the cell C. This algorithm is fairly

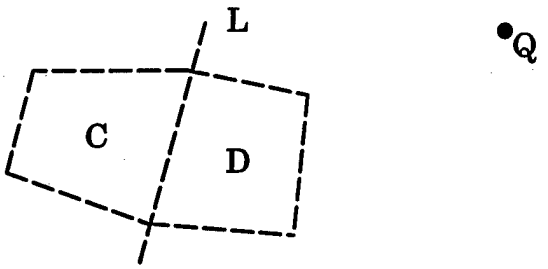


Figure 2.8 Mastins Interpolation Scheme II

robust even on skewed grids. However, in a three dimensional grid, the cell faces can be non–planar. The paper notes that due to this, extension of this algorithm to 3 dimensional grids does not guarantee that the algorithm will terminate at a cell C which contains the point P.

The best suited and the fastest algorithm used in this code takes advantage of the local physical properties by mapping the global geometry in terms of local coordinates α, β, γ of the current cell [20]. The interpolation point x_p is expressed in terms of these local coordinates which fall within a known interval (–1 to 1) if x_p lies within the cell. If any of the coordinates fall outside this

interval, then next cell is guessed based on the value of those local coordinates. The next guess cell can be diagonal to the current cell if more than one coordinates falls outside the interval. Further if any of the coordinate is extremely large (say $\alpha >> 2$) then the algorithm can intelligently jump more than one cell in the appropriate direction. If we assume the nearby cells to have approximately similar aspect ratio as the current cell, then an $\alpha = 20$ would mean that the point x_p lies approximately 10 cells away. This method thus dramatically speeds up the search process.

The coordinates α, β, γ for the point $\underline{x}_p$ are obtained by solving a set of 3 linear equations in terms of the 8 cell vertices,

$$\underline{x}_p = \sum_{j=1}^{8} \lambda_j \underline{x}_j \qquad (2.16)$$

where,

$\lambda_1 = (1-\alpha)(1-\beta)(1-\gamma)/8$ $\quad$ $\lambda_2 = (1+\alpha)(1-\beta)(1-\gamma)/8$

$\lambda_3 = (1+\alpha)(1+\beta)(1-\gamma)/8$ $\quad$ $\lambda_4 = (1-\alpha)(1+\beta)(1-\gamma)/8$

$\lambda_5 = (1-\alpha)(1-\beta)(1+\gamma)/8$ $\quad$ $\lambda_6 = (1-\alpha)(1+\beta)(1+\gamma)/8$

$\lambda_7 = (1+\alpha)(1+\beta)(1+\gamma)/8$ $\quad$ $\lambda_4 = (1+\alpha)(1-\beta)(1+\gamma)/8$

Equation (2.16) can be solved directly by inverting using Cramer's Rule. However, if the point is outside the cell, then a direct solution can give wrong direction sense. As shown in the Figure 2.9 , the sense of direction changes for β beyond the point of intersection of the projection of the two edges. A direct solution would give this exact value and lead to a wrong next cell guess. The paper suggests Newtons method be used instead (Equation (2.17)). The first guess of Newtons method is guaranteed to give the correct directional sense for the next guess cell.

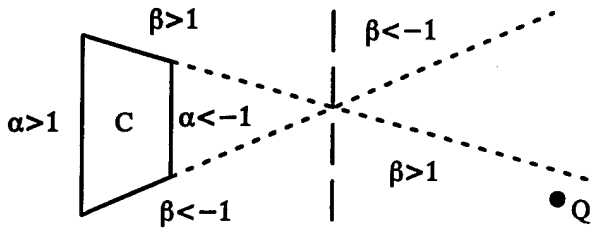


Figure 2.9 Problem with direct solution

$$\underline{f}(\alpha,\beta,\gamma) = \sum_{j=1}^{8} \lambda_j \underline{x}_j - \underline{x}_p$$

$$(\alpha,\beta,\gamma)^{n+1} = (\alpha,\beta,\gamma)^n - \frac{\underline{f}^n}{\underline{f}'} \qquad (2.17)$$

If the initial guess is too far from the location of $\underline{x}_p$, the algorithm can get stuck in local minima if the grid is highly skewed. In the current application, the possibility of the initial guess being too far away from the correct location is very remote, since grid adaptation is not expected to move the points too far from their original location. Further as the search proceeds the first guess of α, β, γ for Newton's iteration in the new cell is made intelligently from the values of the coordinates computed in the earlier cell. In terms of the local coordinates, the width of each cell is 2. Hence going to the next cell should change the value of the associated parameter by approximately 2. This new improvement in the algorithm also greatly improves convergence of the Newton's iteration.

However in case of multiblock solution interpolation, it is still possible that the first guess of a cell location is far from the point being searched. In order to care of such cases, the original algorithm was improved by adding a random jump to pull out the algorithm in case it gets stuck in such local mini-

ma. The random jump also enables the new algorithm to search grids with embedded objects

The search continues in this manner until $|\alpha, \beta, \gamma| < 1$ or till the search hits the boundary and the local coordinates indicate that the point lies outside the domain. If a cell containing the point is found, the local coordinates α, β, γ can be used directly as weights for trilinear interpolation of the solution from cell vertices. This another major advantage of this method. Figure 2.10 shows the solution interpolated to a 2 block adapted grid. The contours match almost perfectly validating the algorithm used.

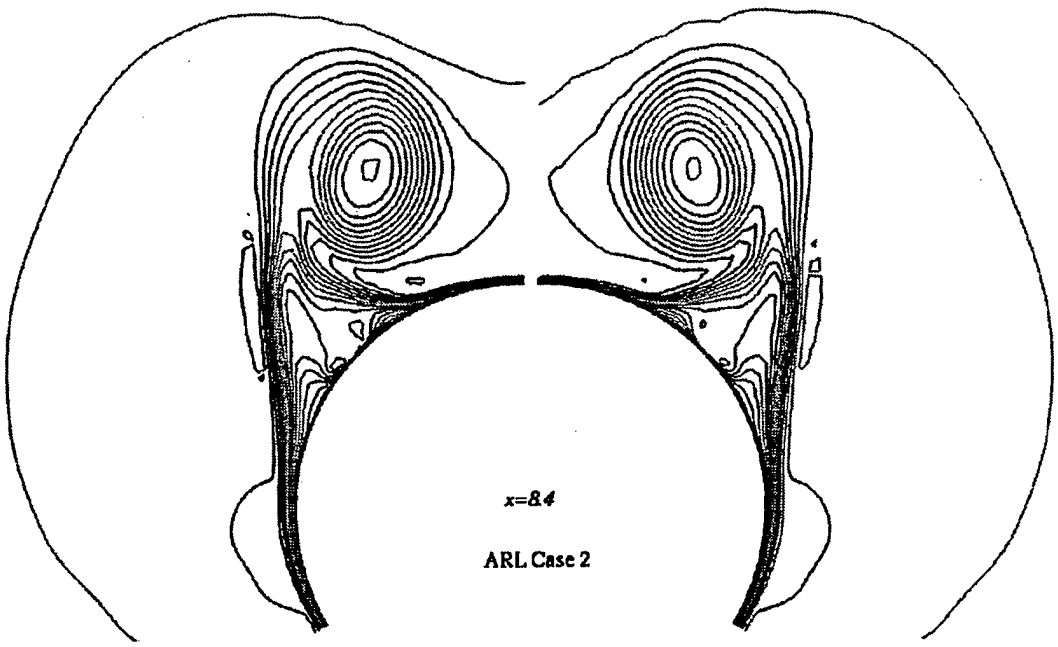


Figure 2.10 Multiblock Solution Interpolation

The aforestated discussion about the search and interpolation considered a single block of grid. Additional complexity is introduced when the domain consists of multiple blocks, and points can move from the domain of one block to another. The blocks then have to query neighbouring blocks for solu-

tion information. This aspect is discussed in more detail in the next chapter on parallel multiblock treatement.

CHAPTER III

PARALLEL MULTIBLOCK GRID ADAPTATION

Background on Parallel Computing

Large computational tasks can be done much faster by using multiple processors. This chapter discusses various issues involved in parallel multiblock grid generation. The current section gives a background on the terminology used in parallel computing. The later sections discuss particular problems associated with concurrent execution of the grid adaptation code and discuss the approach taken to solve them.

For a given level of technology, parallel computers would always have higher computational potential than the vector computers. This comes from a fundament rule in VLSI complexity theory (The AT^2 result). Given an area n^2A for designing a computer, the one with n^2 slower components of size A each is potentially n times faster than a single fast computer of size n^2A. However, to exploit this potential power, the algorithm running on the machine should have n concurrent tasks that are fairly decoupled. In most practical cases, these tasks are not completely independent and need to communicate with each other by the means of messages. The goal in designing a parallel algorithm is to design a set of concurrent tasks that can operate with minimum inter communication, without losing solution accuracy [25].

We shall now take a brief look at the design of a parallel algorithm followed by a discussion of some performance metrics and general terminology used in parallel computing.

Design Methodology

Design of a parallel algorithm can be done systematically in four stages Partitioning, Communication, Agglomeration and Mapping [25]. In the first two stages attention is focused on discovering the best algorithm for maximum concurrency and scalability while the later steps involve locality and performance related issues.

Parititioning involves decomposition of the problem into multiple tasks. Parallel algorithms can be designed using functional decomposition or domain decomposition. Functional decomposition involves concurrent execution of parts of an algorithm that are fairly disjoint. Domain decomposition on the other hand involves partitioning of the data into smaller domains. Each of the multiple tasks work on their particular subdomain. Multiblock grids give a pre–partitioned domain, and hence the domain decomposition approach comes naturally in case of parallel multiblock grid generation

The next step involved in the design of a parallel algorithm is communication. Individual tasks need to communicate the data required for computation. In our case, this data consists of points near the block interfaces, solution maximas and norms, etc. Communication requirements could be local/ global, structured/ unstructured, static/ dynamic, and synchronous/ asynchronous. Allowing a general multiblock topology means we can have arbitrary graphs representing block connectivities. Hence communication pattern is unstructured. Exchange of face information is thus an example of local unstructured static asynchronous communication. Caculation of norm at the end of each iteration on the other hand is global synchronous communication. The message passing interface chosen for this program must therefore support asynchronous communications.

Agglomeration looks at actual organization of the algorithm with attention focussed on increasing algorithmic efficiency. This would include grouping of particular tasks, and replication of the data and associated computation to minimize time lost in communication. In a one dimensional elliptic solver, each point has a three point stencil. This means that a point i needs information from points $i-1$ and $i+1$ for the computation. For points on block interfaces, this would mean that each point will have to query the next block for information about its neighbour. This communication overhead is easily removed by replicating the plane adjacent to the block interface belonging to the neibouring block. The next section talks about this in greater detail.

The final step of mapping considers where each process is executed. Mapping adjacent blocks on adjacent processors can give higher communication efficiency. This would require knowledge of the system hardware which is beyond the scope of this work. However, the underlying message passing interface that spawns and manages the processes can provide such functionality (MPI) [27]. Provision has been made in the code to utilize that feature if it is provided. The algorithm also provides an option for the user to manually control the processors to which each process maps on SGI machines.

Performance Metrics

Performance measurement of a parallel program can consist of a variety of considerations. Execution time, memory requirements, parallel effiencicy, latency, I/O rates, design costs, maintainance costs, portability, hardware requirements, hardware costs, potential for reuse, scalability etc. are just to name a few. The most important metrics vary for each application [25].

Minimization of execution time on the available hardware is the basic goal of this algorithm. This can be achieved if the code makes the best use of available computational power.

Efficiency is a measure of the fraction of time that processors spend in doing useful work. It characterizes the effectiveness with which an algorithm uses the computational resources of a parallel computer in a way that is independent of problem size. Efficiency can be formally defined as

$$E_{relative} = \frac{T_1}{PT_p} \tag{3.1}$$

where T_1 is the execution time on one processor while T_p is the execution time on P processors [10][25].

Speedup gives the reduction in execution time achieved by running on P processors.

$$S_{relative} = \frac{T_1}{T_p} = PE_{relative} \tag{3.2}$$

In case of the parallel multiblock solver discussed here. The number of processes depend on the number of blocks in the grid. Execution time on a single processor is thus, impossible to find. We can however define the efficiency metric with respect to the execution time of a representative sequential multiblock grid generator. Table 4.5 in Chapter IV compares run time for a five block grid grid with that using GUMB.

Scalability is an important performance measure that describes behaviour of an algorithm with changes in the problem size or the computational power. The exact definition of scalability is highly debated [10]. One of the most widely accepted methods is to define scalability in the sense of isoefficiency. The isoefficiency function is defined as the rate at which the problem size should grow with the number of processors to maintain a fixed effieciency.

PMAG is primarily aimed at aerospace applications. The test cases available have a fixed problem size in terms of number of grid points and number of blocks. Thus, obtaining a performance metric in terms of isoefficiency is also very difficult. The results presented in Chapter IV compare the time required to adapt grids to the flow around a missile using different number of blocks. There is also a comparison of the times required to run 6 blocks on different number of processors. While niether of these give a precise value for the scalability of the algorithm, they can be viewed as qualitative results indicating the potential.

Parallel Implementation

With that background on Parallel algorithm design methodology and the important metrics for assesment of its perfomance, this section takes a look at the actual implementation details of the parallel grid generation algorithm.

Each block has to be solved in an independent processes. PMAG spawns processes equal to the number of blocks in the topology. Each block is expected to be stored in a separate disk file. This is done to enable concurrent reading and writting of grid files by each process. The user needs to supply the connectivity information for each block. This includes information about shared faces and fixed patches (Figure 3.2). The grid blocks are read concurrently by all the processes.

Grid lines must be continuous across adjoining blocks. The inter–block faces, edges and vertices which do not describe a definite fixed body must be free to float in the space. This requires that the block faces be solved with an exchange of information across the block faces. A strategy of replication of

data is adopted to minimize the communication which is expected to be costlier than the extra computation.

Once the grid is read in, the blocks add ghost faces to accomodate grid points from neighboring blocks to maintain continuity across shared faces. Exchange of data is done in an efficient manner by the use of asynchronous (non–blocking) communication calls.

The processor first posts recieves for all patches expected and then packs and sends the patches from itself to its neighbors. In the meanwhile some of the recieves are completed in the background. The recieved buffers are then copied out into appropriate buffers as the calls are completed. This technique effectively minimizes the time lost in communication.

Each processor then computes the control functions and runs the elliptic solver. Face information is exchanged at the end of every iteration of the elliptic solver and a global norm is computed to determine the convergence criteria. Solution interpolation and boundary point movement using NURBS is done after every n iterations as specified by the user.

It might be noted however that the number of tasks is fixed by the number of blocks in the grid. If the algorithm is restricted to this, it would not be scalable in terms of its ability to solve larger problems on more number of processors. This algorithm achieves scalability by the use of threads. If more processors are available, the processes subdivide their domain by unrolling the outmost loop of the solver and the search algorithm. Threads are spawned to work on each subdomain.

Use of threads is advantageous in more than one ways. Scalability is the primary advantage. Maximum use of available resources is harnessed by splitting blocks into threads. Controlling number of threads spawned by each

process aids in load balancing. The larger blocks can spawn more threads than the smaller blocks. Splitting the domain into smaller chunks leads to better cache performance.

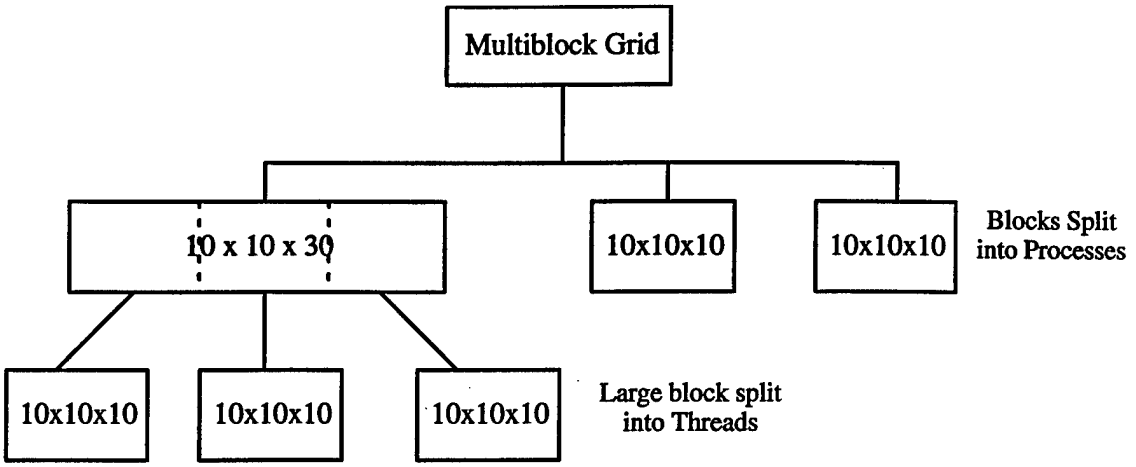


Figure 3.1 Splitting a block using threads

The disadvantage of this approach is that it assumes a shared memory machine capable of creating Light Weight Processes (LWP). In case the code has to be run on a distributed memory machine, or a network of workstations, then a preprocessor splits the grid into appropriate number of blocks before spawning the grid adaptation processes through MPI.

On SGI machines, the algorithm allows the user to specify the processor/processors on which the block and its associated threads will be solved. This feature helps in load–balancing in case the number of processors available are less than the number of blocks. The user can specify the large blocks to be solved on individual processors and the smaller blocks to be grouped together to share the rest of the processors.

The sample input file for an individual block shows the way a user can specify the threads to run, the shared faces, fixed boundaries, and NURBS surfaces

```
3                                   Number of THREADS
1                                   Specify CPUs ? (1/0)
  0 1 3 4                           List of CPUs if line 2 = 1
0                                   Imin has 0 shared patches.
3                                   Imax 2 has 3 shared patches
  1,1,29,1,64                       block,jmin,jmax,kmin,kmax
   2,41,71,1,64
   3,1,70,62,97
0                                   Jmin has 0 patches
0                                   Jmax has 0 patches
0                                   Kmin has 0 patches
0                                   Kmax has 0 patches
1                                   Any Fixed Patches? (1/0)
1                                   0 Fixed Patches on Imin
 1,1,70,1,97                        NURBS,jmin,jmax,kmin,kmax
1                                   1 Fixed Patch on Imax
 0,30,40,1,63                       NURBS?,jmin,jmax,kmin,kmax
0                                   0 Fixed Patches on Jmin
0                                   0 Fixed Patches on Jmax
0                                   0 Fixed Patches on Kmin
0                                   0 Fixed Patches on Kmax

NOTE: Faces must be specified in cyclic order. for example
a J=constant face is specified as kmin,kmax,imin,imax
```

Figure 3.2 Sample Block Information File

The next figure shows the flowchart for the parallel grid adaptation algorithm.

Treatment of Shared Faces

One of the most important factors in multiblock grid generation is to allow movement of points on the shared block faces. These blockfaces in space are fictitious surfaces and hence should not be imposed as fixed boundaries.

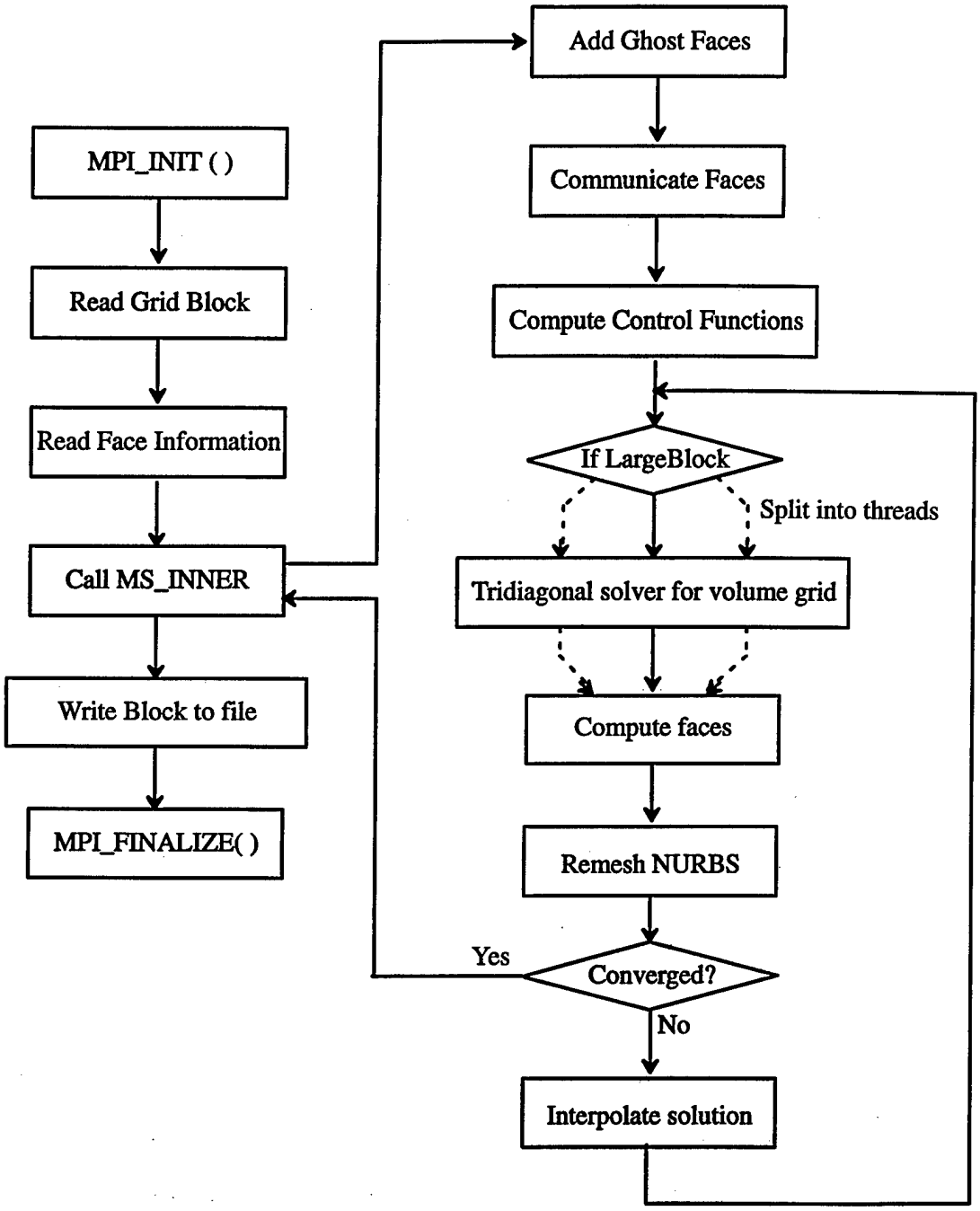


Figure 3.3 Flowchart

Further it is important that grid lines have C^0 and C^1 continuity across the grid faces.

To guarantee such a continuity, at the end of each iteration, each block sends a face to its neighbouring block as shown in Figure 3.4 . The elliptic generator can then run using the face from the neighbouring block as the dirichlet boundary. After every iteration the new updated faces are transmitted to the neighboring block. This way , the face points are solved using the elliptic equations at every step. Each block solves the for the face points separately. Hence it is possible that the face points may not coincide after an iteration.

To get rid of this slight discontinuity, some sequential multiblock codes keep a track of faces, edges and vertices belonging to each block. Connectivity tables are maintained to identify shared vertices edges and faces. At the end of every iteration, these connectivity tables are used to pick out the copies of the common face,edge or vertex and then an averaged (or elliptically solved) value is broadcast to all the owners. Another method especially amenable in C codes uses pointers to faces, so that only a single copy of a shared face is maintained, with both blocks pointing to the same memory location.

Clearly the later strategy cannot be used when the blocks are being solved on different processors. The first strategy of collecting all copies and then broadcasting an average can work in a parallel implementation but it means a lot of communications. (8 vertices 12 edges and 6 faces per block).

Assume a simple structured topology. Each face can be shared by only one other block. One edge is shared by 4 blocks and a vertex is shared by 8 blocks. In such a case, for getting unique face,edge and vertex locations Each block needs to communicate with 26 neighbouring blocks!

Using the above strategy would require a master process to maintain the entire connectivity information . This process would then coordinate the communications and averaging of faces. Such an algorithm would essentially consist of large sequential part when the master coordinates the communications. Further complicated connectivity tables will have to be maintained. Hence attention was diverted to invistigating an alternative algorithm which would substantially reduce the complexity.

It was argued that since elliptic system has a 3 point stencil, and since both the blocks have identical copies of all the three faces required for computation of the block boundary face, the face points calculated by the each block using the elliptic system should be the same.

However, problems arise if the grid generation algorithm uses a tridiagonal solver to compute the new location of the points. Since the tridiagonal solver consists of a forward and backward sweep along a coordinate line (say the I direction), The new location of any point (i,j,k) is no longer dependent only on (i–1,j,k) and (i+1,j,k). This leads to major discontinuity of lines on the I= minimum and I = maximum boundary faces. (J and K faces will not have a problem)

A different approach is taken to guarantee that this does not happen. While the block interior is solved using the tridiagonal system (since it gives faster convergence), the faces are solved individually using Point Jacobi Iteration. The control functions for the points on the shared face are also evaluated using only a 3 point stencil guaranteeing that neighboring blocks compute exactly identical locations of the shared points .

This approach is in essense similar to the averaging approach described above. However, in this strategy each block computes the shared faces, instead

of a single processes computing the shared points and then broadcasting them to all owners. This strategy avoids communication by doing redundant computation on each processor.

In the strategy described above, one block needs to communicate twice (see next section.) with only 6 blocks, thus requiring 12 communications. So even for a simple topology this strategy cuts down the communication by one half. The advantage of reducing communication becomes even more important for a general topology with N blocks sharing a face,edge or vertex.

Simplicity of the entire scheme is a major advantage. No complicated global connectivity tables need to be maintained. This makes the code extremely flexible, enabling it to handle a wide variety of block topologies, including 0 grids embeded in H grids, Periodic boundary conditions etc.

For the above strategy to work, the basic premise is that all blocks sharing a particular point must have identical copies of its complete stencil. For each block, the user specifies the blocks that share a face with it. This means that a block has no information about its diagonally opposite neighbour. A communication strategy that allows each block to acquire information from its diagonal neighbours is described in the next section.

Block–block communication of shared faces

The approach described above requires that a block knows the extra faces entirely. If blocks send/recv only their own faces, then it gives rise to a problem as shown in the figure for a 2–D case.

To overcome this problem, the blocks communicate faces inclusive of the extra points recieved from the nieghbouring blocks (shown by the dashed–blocks). This however means that the block/blocks that send the points before

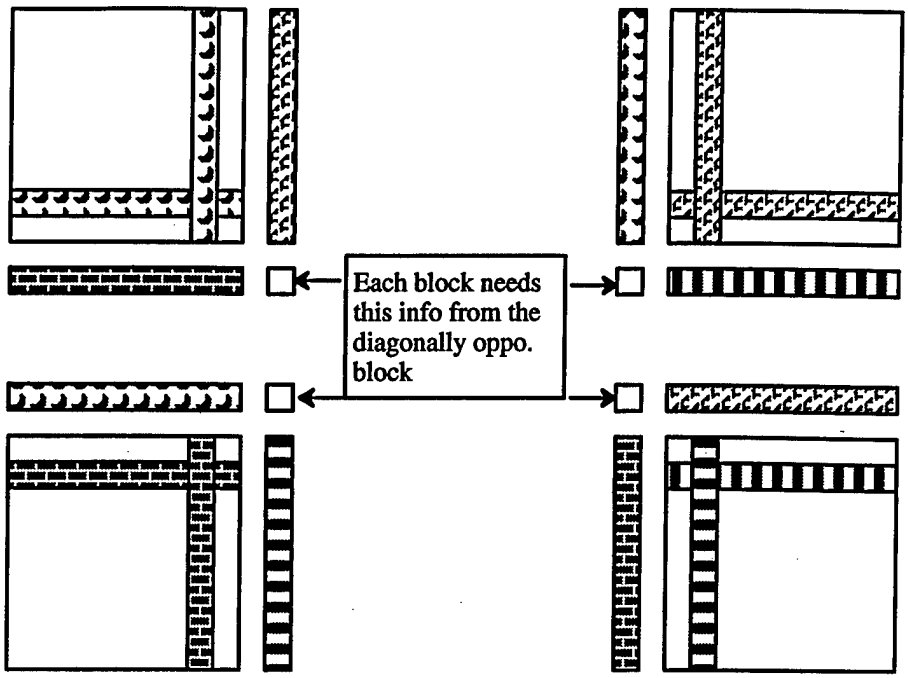


Figure 3.4 Problem in conventional face exchange

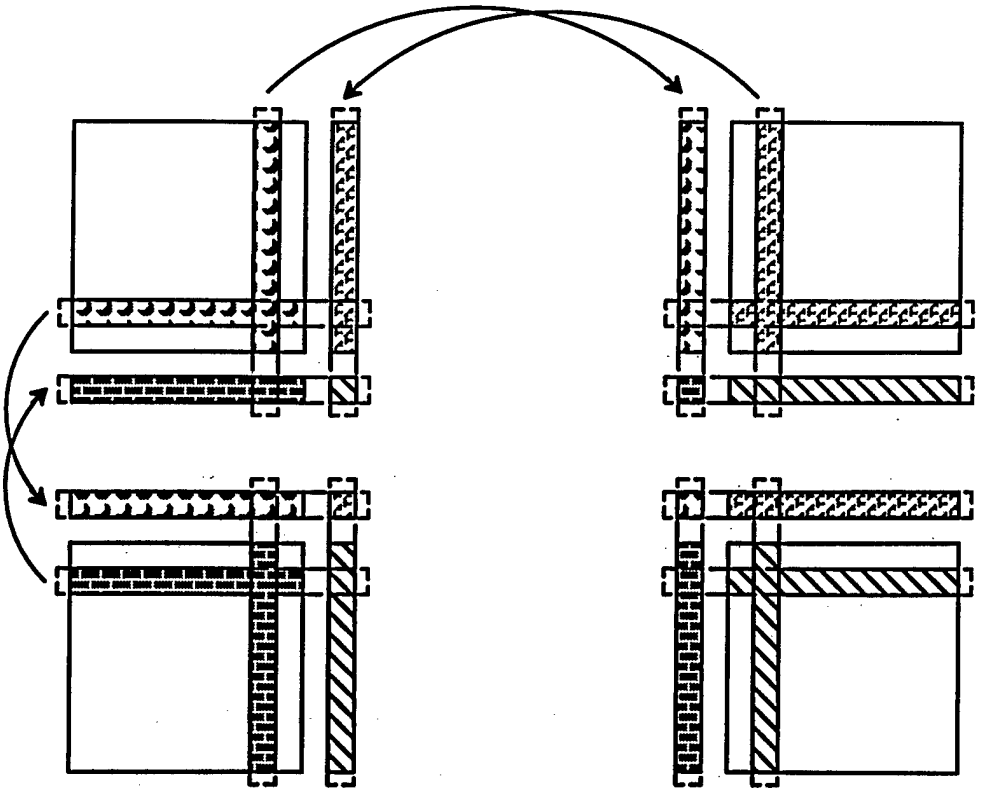


Figure 3.5 Strategy to get corner data

recieving the faces from the nieghbors would send out void/old points to the neighbours

One way to over come this problem is to have the communication go in a cyclic loop. So that each face communicates its information only after it has recieved the relevant information from its other neighbours. This however means that the entire communication will be sequential and hence a great increase in the communication time. Further the blocks taking part in a communication need to know all the neighbours. So its necessary to store the connectivity information of the all the blocks. This defeats one of the major advantages gained in the face treatment strategy.

The problem can be effectively overcome if we do the same communication one more time to fill up the voids. This strategy allows us to use asynchronous communication, so that more than just 2 blocks communicate at any instant. Further no additional connectivity information needs to be stored. Each block needs to only know the nieghbours that share its faces. Note that doing the entire communication twice does involve trasmission of some redundant information. This could be avoided by sending only the edges and vertices in the second communication. This has feature has not been implemented in PMAG yet.

This strategy described in this section is unique to this work. The next section discusses solution interpolation algorithm for multiple blocks running concurrently.

Parallel Multiblock Solution Interpolation

The solution adaptive grid generation process involves moving grid points according to the solution gradients. However as the points are moved the original solution needs to be interpolated on to the new grid. A parallel

gird adaptation algorithm supporting general multiblock topologies makes solution interpolation significantly more complex. The grids points can move outside the original domain of a block . In such a case, the block does not have enough information to interpolate the solution and adaptive functions to all its points. Each block now needs to query all other blocks for the points that are no longer within its own domain.

The code presented has been designed to required minimal dependency on topological information. Since each block knows only its immediate neighbours that share a face, it is impossible to intelligently query other blocks for solution information. (The point may have moved to a diagonally opposite block of which the current block has no information).

The strategy in this case is to involve all the blocks in the query. The information can be queried from the other blocks by either passing information of extra points thru the list of blocks in a circle (Shift), or by concatenating all the points into an array (Allgather) and then doing an Allreduce over all the blocks to get the solutions. The Shift operation takes P steps to complete. While the Allreduce takes 4*log P steps to complete. As a result, for number of processes $P < 16$, the shift is faster than the all reduce. Ideally a polyalgorithm should be used to switch methods according to the process group size. The current implementation however uses only the Allreduce method for the queries.

Figure 3.6 shows the algorithm used for parallel multiblock search and interpolation. The algorithm can be made scalable by splitting each of the for loops over multiple processors using threads.

The next chapter discusses the results obtained on several test cases. It demonstrates the capability of the algorithm to handle diverse configurations, and lists its relative merits/ demerits over other similar packages.

```
For each point in new grid
{
   Search point on old grid.
   if point_not_found  Add point to list of ExternalPoints
   else Interpolate solution to the point location.
}

Gather all external points into AllExternalPoints.

For each point in AllExternalPoints
{
   Search point on old grid,
   if point_not_found     Solution = 0.0
   else  Interpolate solution to the point location.
}

Sum the Solution array across all blocks to make it known
to all blocks.
```

Figure 3.6 Algorithm for Parallel Multiblock Search

CHAPTER IV

RESULTS

The algorithm PMAG described in the earlier chapters has been successfully applied to several test cases. This chapter enlists the results obtained and lists the merits of this code as compared to the pre–existing codes. The last section points at possible future work.

The ARL Missile test cases

The main motivation for generating this parallel adaptation code came from the need for developing a fast grid adaptation algorithm for simulating the ARL Missile test cases [30]. This work was a part of the Computational Fluid Dynamics (CFD) simulations performed in support of the KTA–12 program. This program was designed to evaluate computational technology for predicting highly separated flowfields for missile configurations.

NPARC3D flow solver was used for computing the solution. This solver is based on the Beam–Warming implicit approximate factorization algorithm that solves the set of equations produced by central–differencing the Navier–Stokes equations on a multi–block structured grid. The code is very well documented and validated [18]. The turbulence model used is Baldwin–Lomax.

The existing code by Hugh [2] with the CSIP solver was very slow. Table 4.1 shows the run time for the original algorithm as compared to the time taken for the current algorithm. Figure 4.1 shows the comparison of point clustering near the shock for each algorithm.

Algorithm	*# Blocks*	*# Procs*	*Run Time (real)*	*Speedup*
Hugh's Solver	1	1	7:34:58.29 hours	
PMAG	1	1	58:15.64 mins	1
PMAG	2	2	32:33.70 mins	1.8
PMAG	6	6	13.14.23 mins	4.4

Table 4.1 Comparison of runtime for PMAG with Hugh's Algorithm

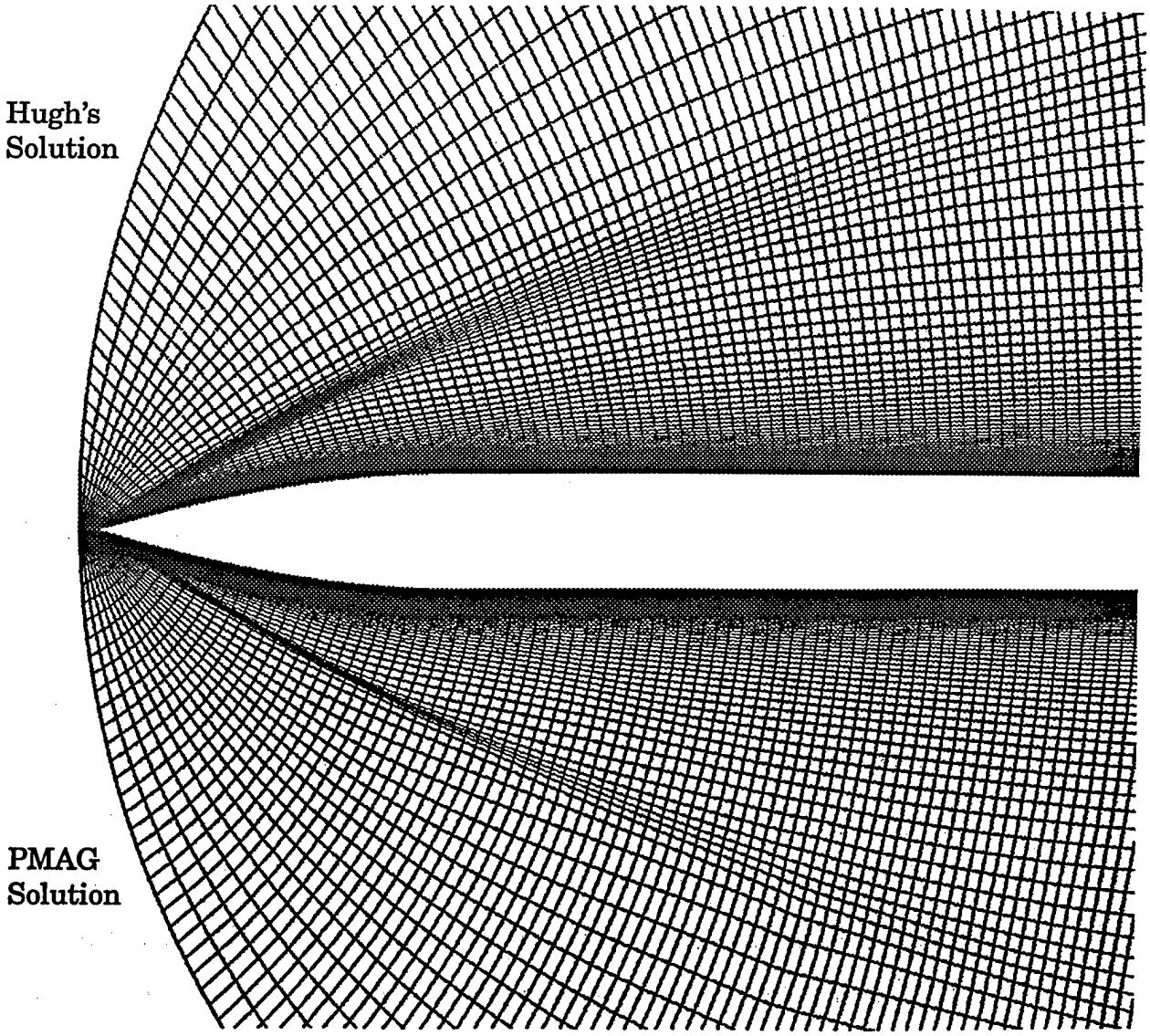


Figure 4.1 Comparison of adaptation at shock

All the runs in Table 4.1 were done on a 12 processor SGI Power Onyx machine (CPU: 150Mhz R4400, Main memory: 1.5GB). For sequential runtime, all the blocks were combined into a single grid block and the same program was used. This result cannot be used for judging the scalability, since the single block run does not involve communication of boundary information between faces, and multiblock solution interpolation. The efficiency of 0.73 inspite of the overheads is thus substantial in this context.

Parallel Performance

Im most cases with complicated multiblock topologies, the grids cannot be combined into a single block. In such a case, if the numer of processors available is less than the number of blocks, the process have to time share on the vailable CPUs. On the other hand if the number of processors exceeds the number of blocks, each block can spawn multiple threads to take advantage of available resources. Table 4.2 shows a comparison of runtimes on a range of processors. The speedup and efficiency is calculated based on the runtime for the entire grid as a single block (Fastest sequential runtime possible).

# procs	Run Time (real)	Speedup	Efficiency
1	1:12:20:38 (expected)	0.8053	
2	40:20.38 mins	1.4442	0.7221
3	26:30.06 mins	2.1981	0.7326
6	13:14.23 mins	4.4017	0.7336
9	11:28.21 mins	5.0799	0.5644
11	10:07.67 mins	5.7570	0.5227

Table 4.2 Absolute Speedup and Efficency for 6 blocks

PMAG can also be run over a network of workstations. Table 4.3 shows the runtime for a cluster of SGI Indigo^2 workstations. Each workstation has a

100MHz R4000 CPU and 128MB main memory. The super linear speedup is because of memory swapping. (The grid is 80MB and solution is 110MB.)

# Blocks	# machines	Run Time (real)	Speedup
1	1	4:02:40.24 hours	1
6	6	31:54:48 mins	7.6

Table 4.3 Runtime over a network of workstations

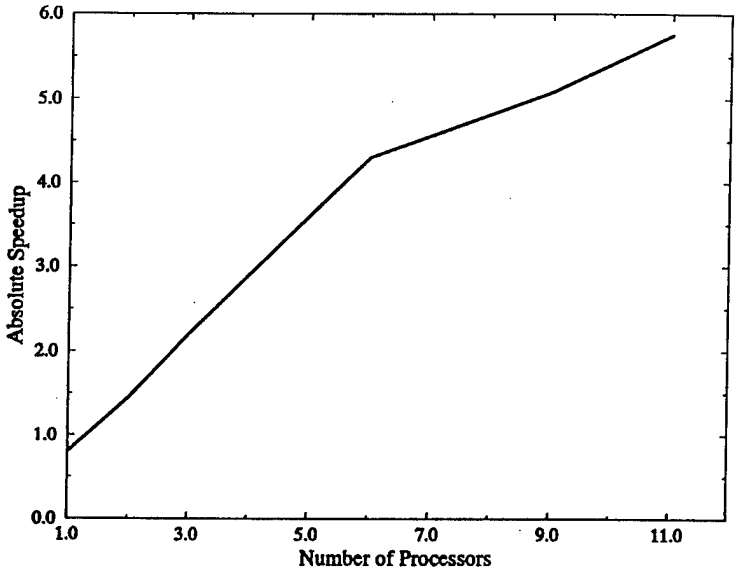


Figure 4.2 Graph of Speedup

Grid Adaptation Results

Grid adaptation was used for 3 test cases in the ARL report [30]. Table 4.4 shows improvement in body force calculation by grid adaptation in case 3. (Mach 2.5, angle of attack 14°, P_0 = 42" Hg, T_0 =308 K, Re =4x10^6 /feet)

The figures on the following pages show the improvement in solution accuracy in terms of better resolved flow features and improved body force computations, obtained for test case 2 (Mach 1.8, angle of attack of 14°, P_0 = 14", Hg, T_0 = 315 K, Re = 2x10^6 /feet)

Forces	Experimental Data	Unadapted NPARC (BL)	Adapted NPARC (BL)
Axial	0.1957	0.3307	0.3309
Normal	1.9100	1.8855	1.9052
Moment	10.2417	10.0314	10.2060

Table 4.4 Forces and Moments Comparison for Case 3

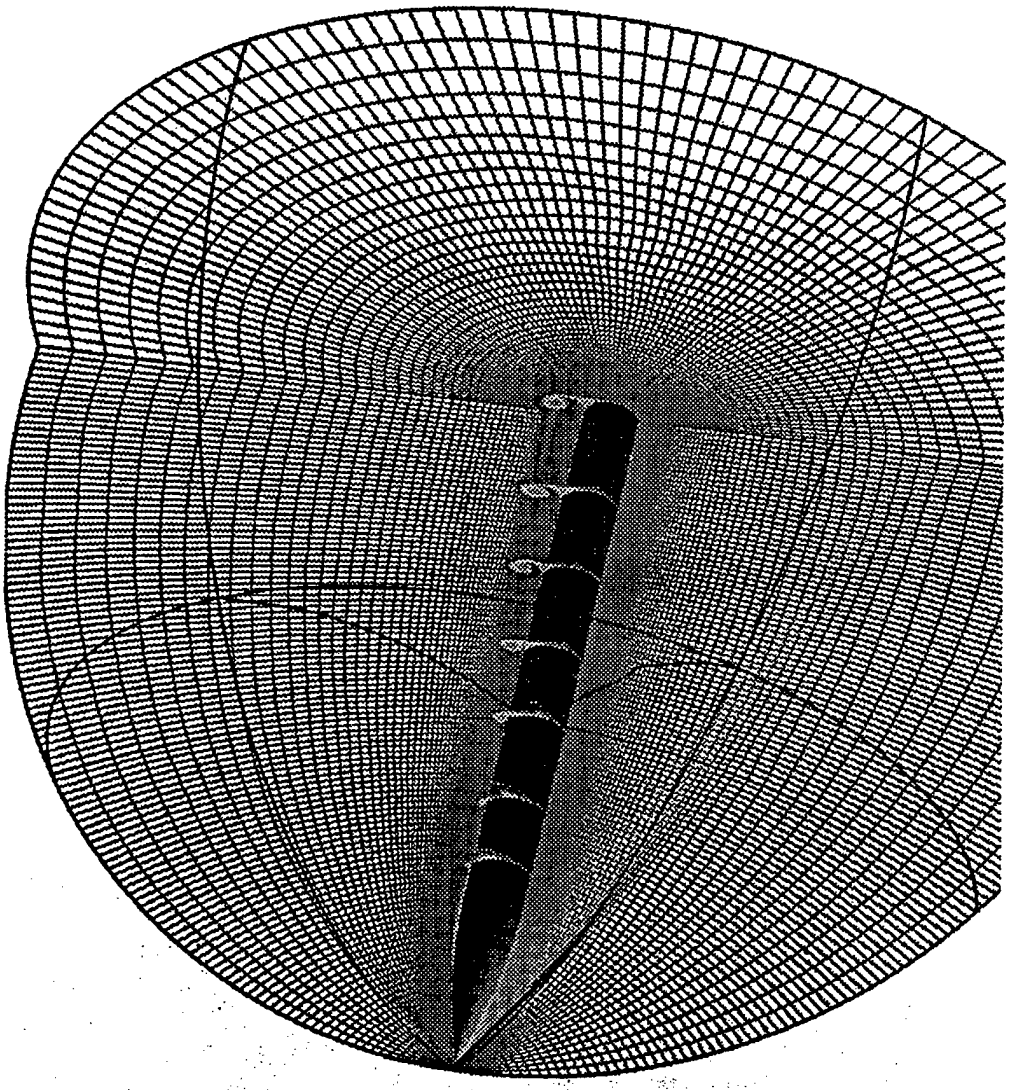


Figure 4.3 6–Block Grid around the Missile

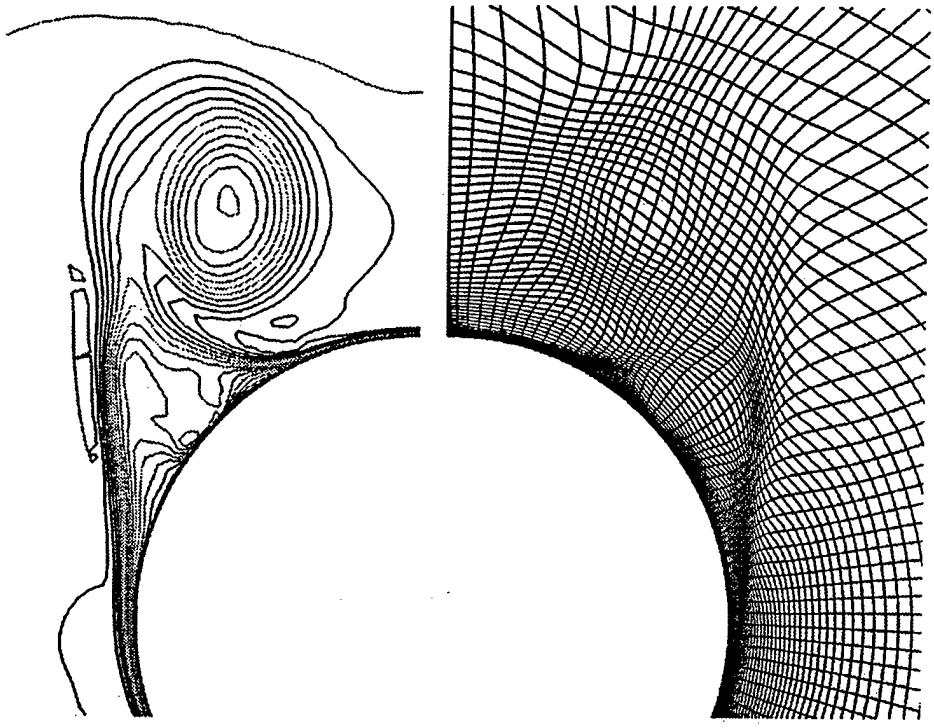


Figure 4.4 Vortex at x=8.4

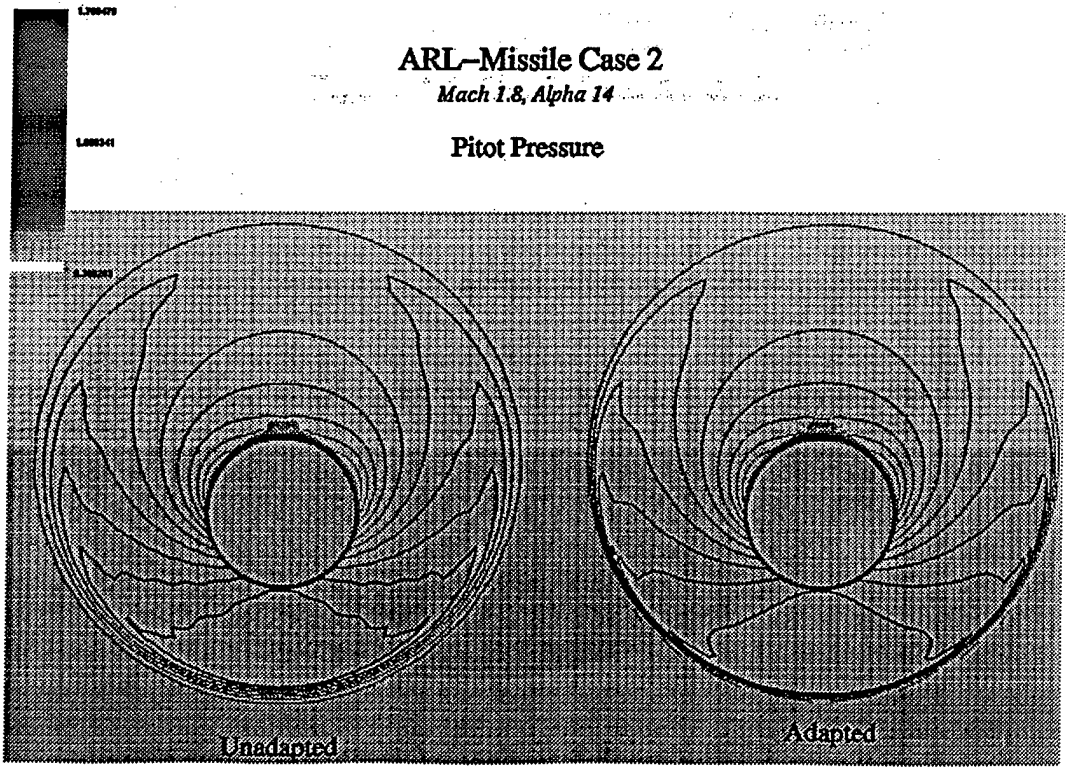


Figure 4.5 Shock from front view

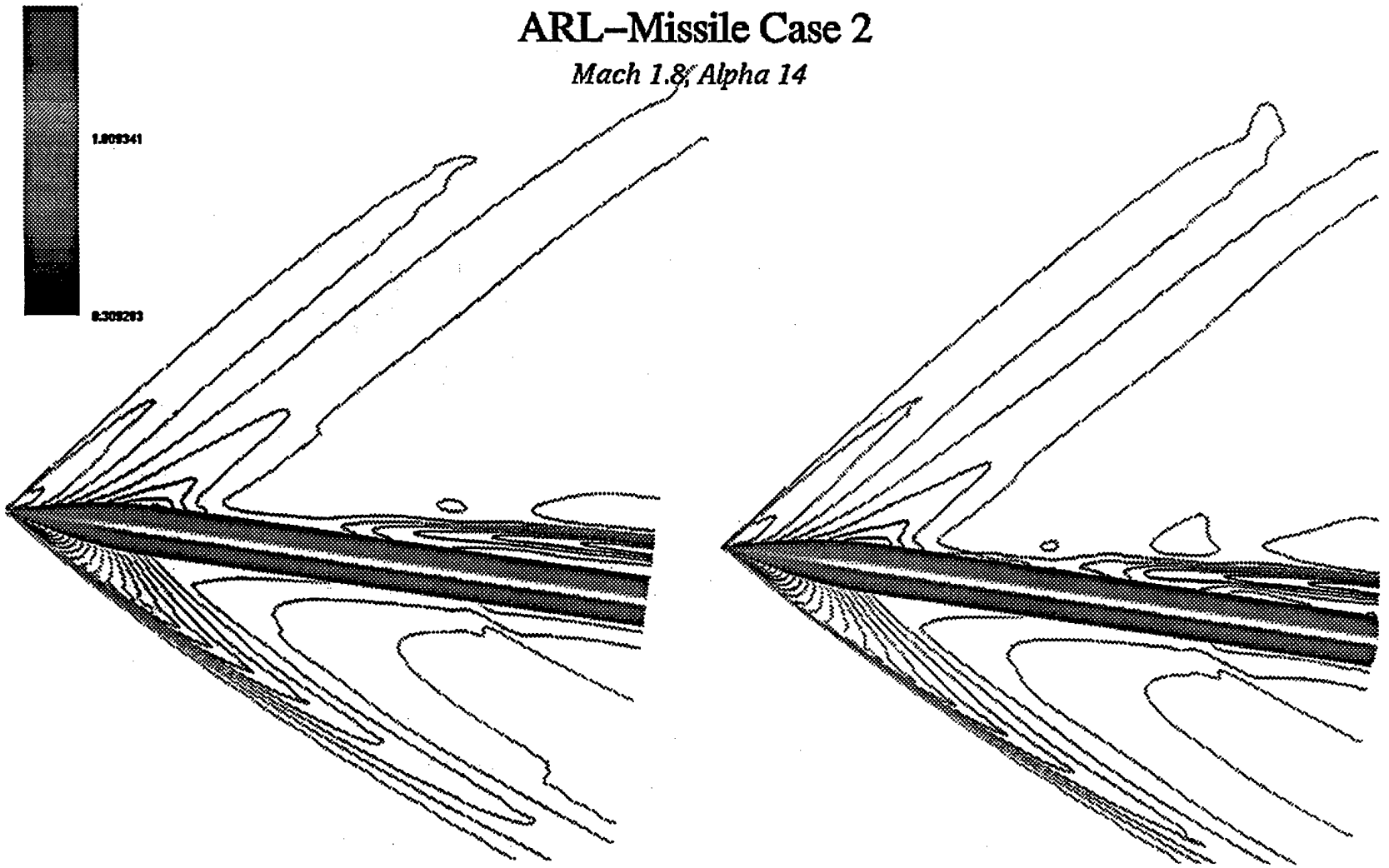


Figure 4.6 Improvement in shock resolution

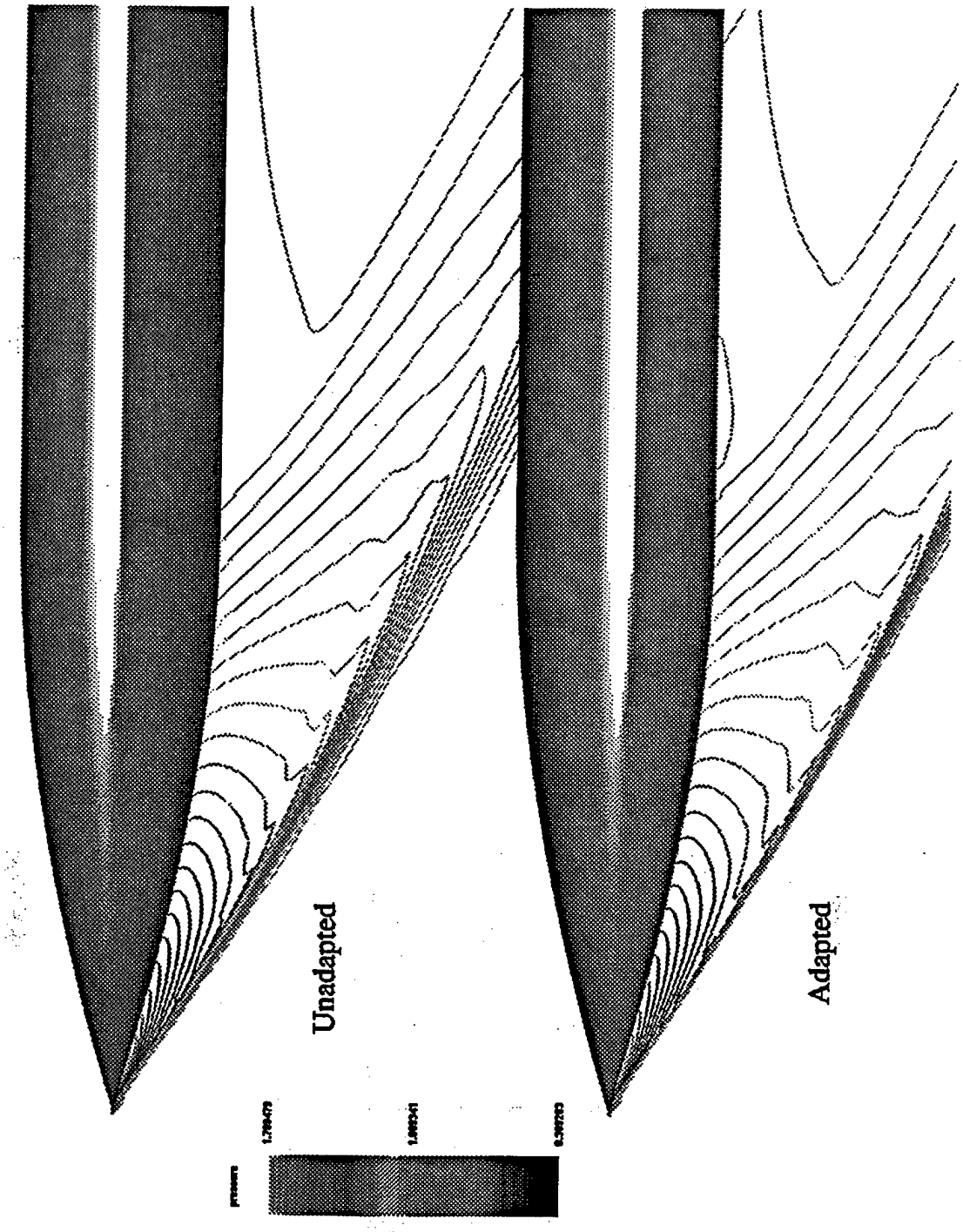


Figure 4.7 Close up of improved shock

5–Block grid around a Missile Fin

Table 4.5 gives the comparison of runtime for generating an elliptic grid in 5 blocks around a missile fin. The results show that a near linear (absolute effieciency 0.96) speedup is achieved by the parallel algorithm. The speedup in this case looks much better than the ARL–Missile cases, since this example does not involve grid adaptation. NURBS evaluations are not done since surfaces are treated as dirichlet boundaries.

Algorithm	*# procs*	*Run Time (real)*	*Speedup*
Sequential (GUMB)	1	2:36:16 hours	1
PMAG	7	23:55 mins	6.72

Table 4.5 Comparison of runtime for GUMB v/s PMAG

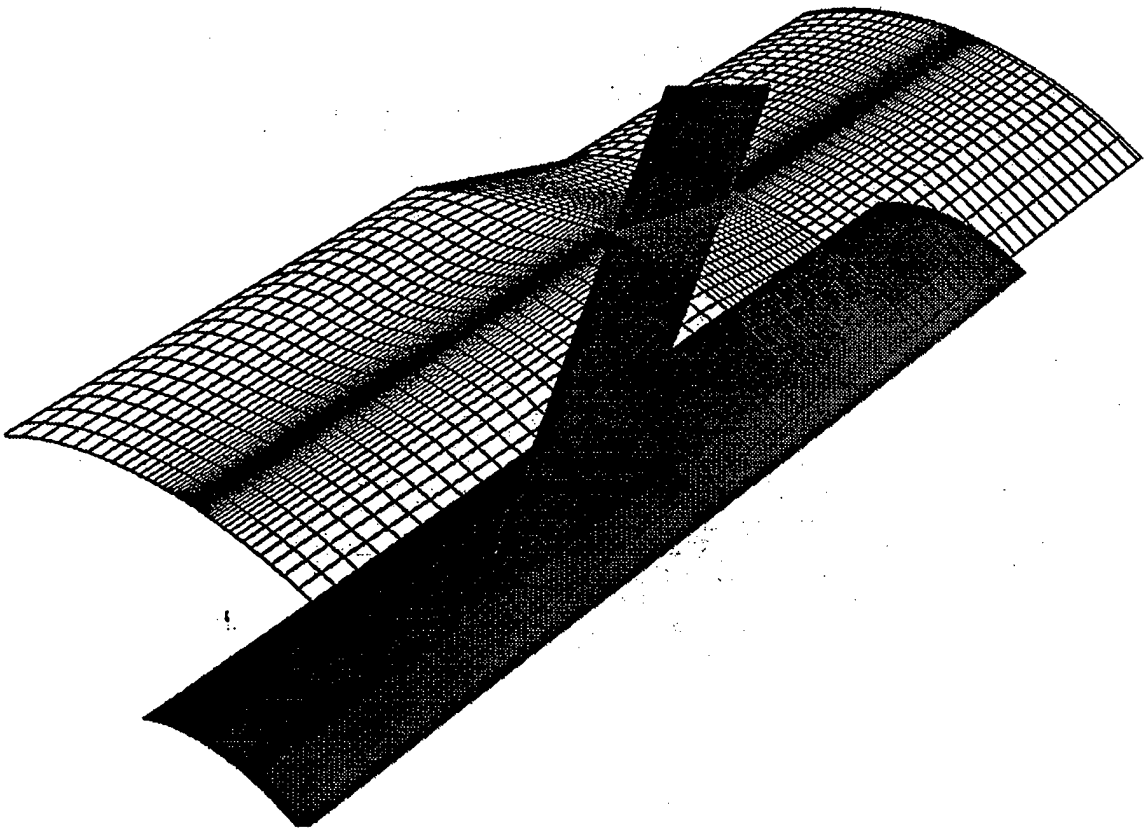


Figure 4.8 Smoothened grid around the fin

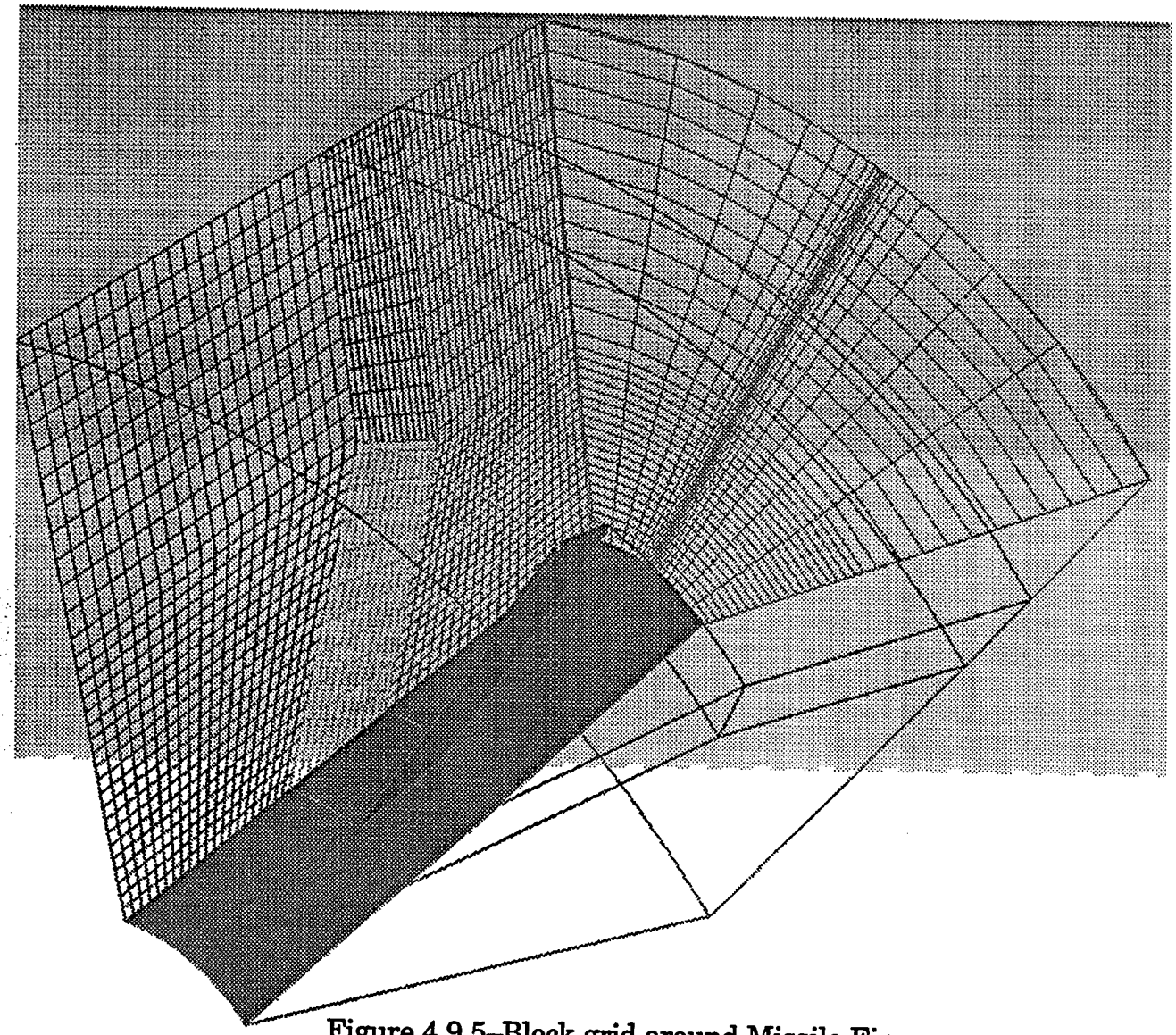


Figure 4.9 5–Block grid around Missile Fin

General 3–D geometry

This example demonstrates PMAG's capability to generate smooth elliptic grids around complicated topologies. The grid is a single block O–grid. K minimum and K–maximum faces coincide with each other. The O–grid is simulated by specifying the K faces to be shared, with the next block as the current block itself. The block then sends the face to itself and puts it in the ghost cells in the appropriate place. When a block communicates with itself, PMAG assigns unique tags to the faces, so that messages do not get mixed up.

Figure 4.13 shows the critical region for this geometry. An elliptic grid would usually crossover at the corner. The grid shown was obtained by doing one smoothing iteration on the geometric control functions evaluated over the entire grid. (No smoothening is done for the control function affecting the packing.) The grid is allowed to converge, and geometric functions are recomputed after a few iterations (in this case 20). The final grid was obtained in 35 iterations.

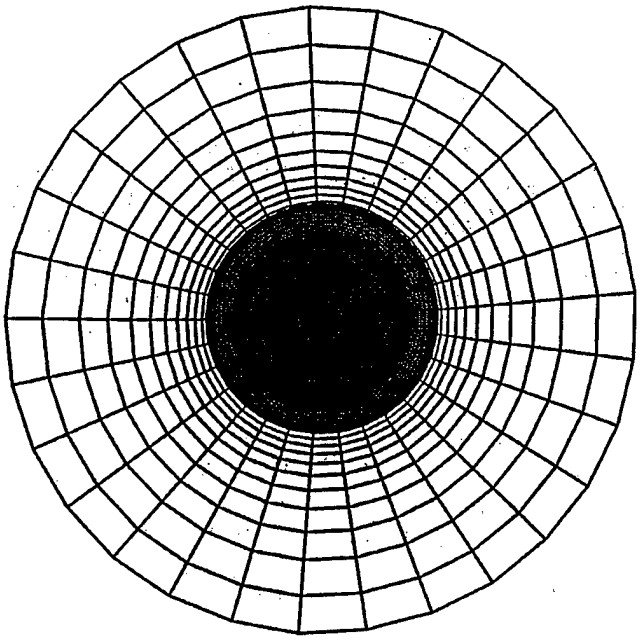


Figure 4.10 The O type grid

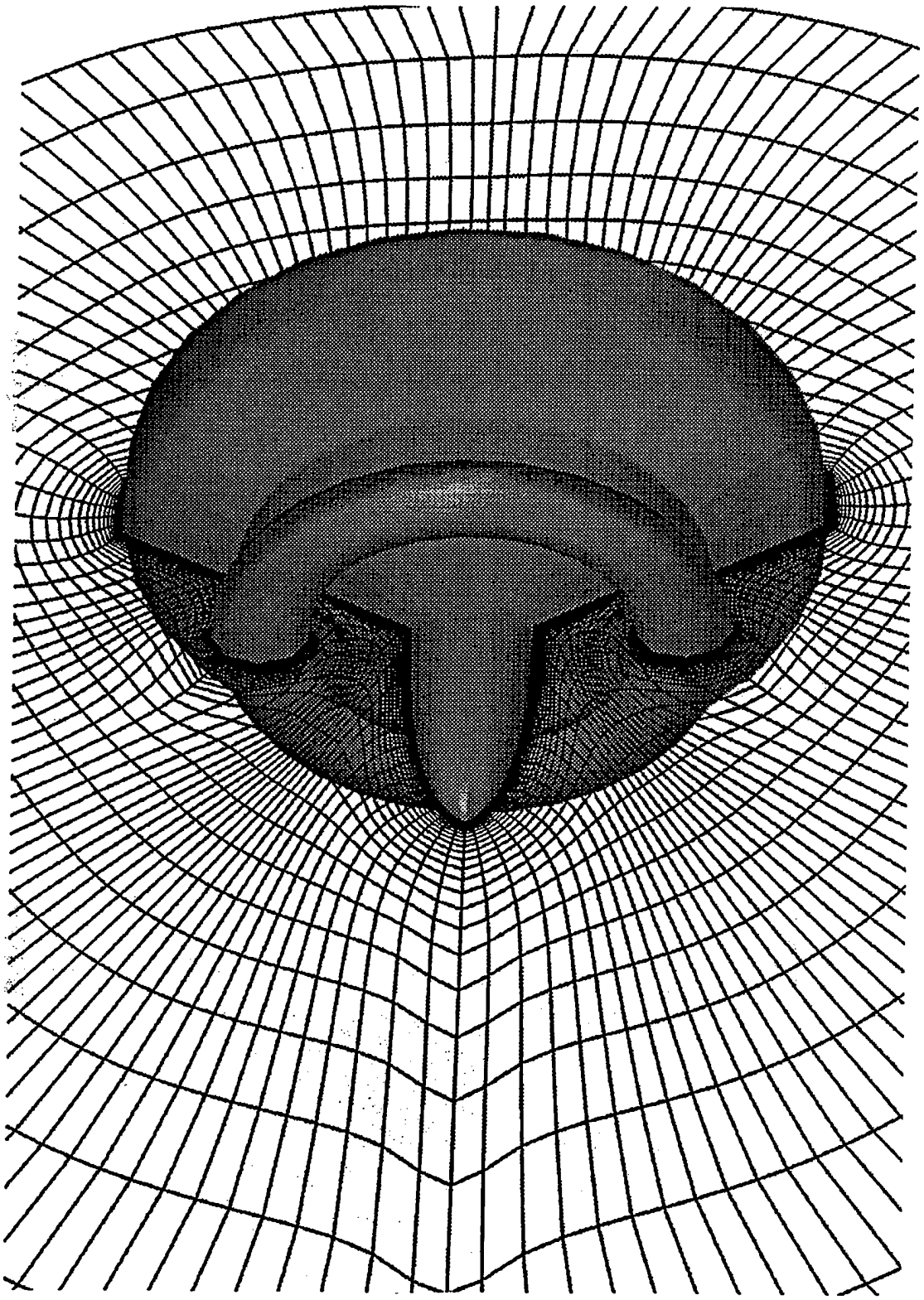


Figure 4.11 3D geometry with grid

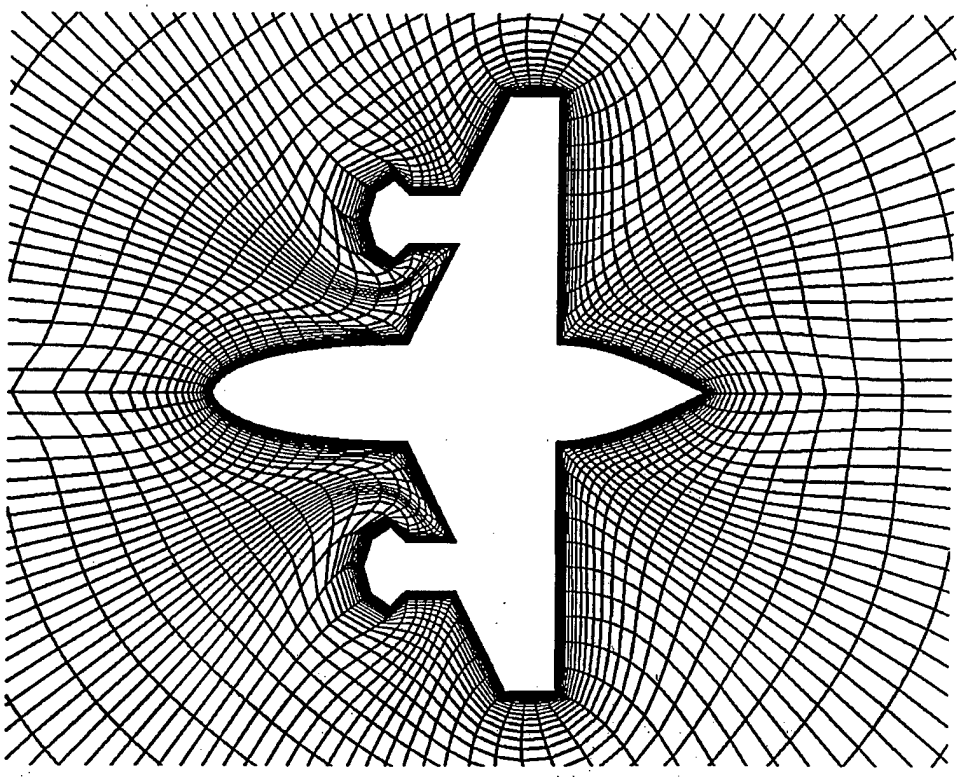


Figure 4.12 Side view of entire geometry

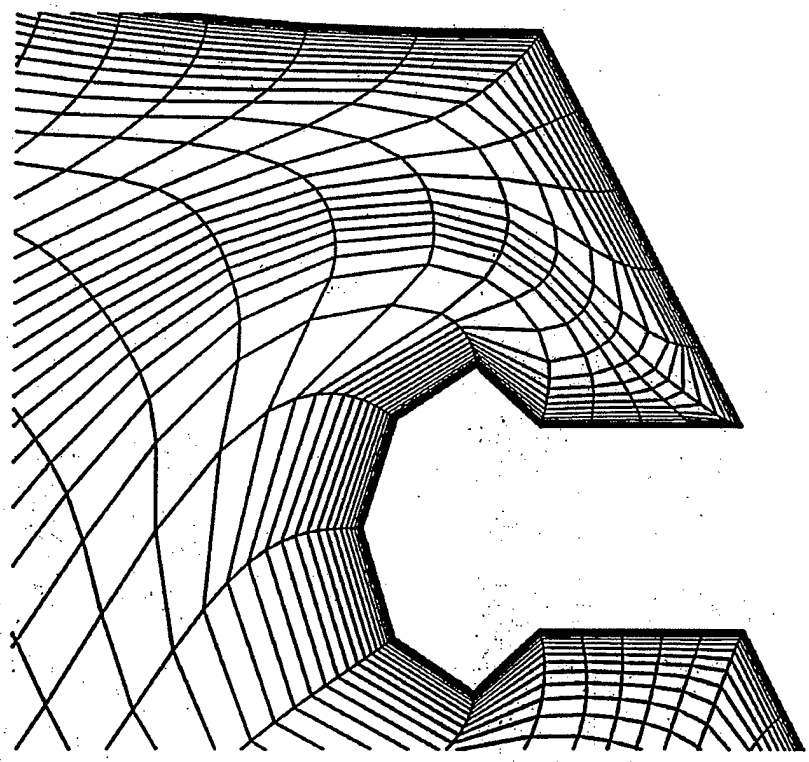


Figure 4.13 Zoomed view of critical section

Bronchial Tubes

The following example shows multiblock grids used in biomedical applications. The grids represent bronchial tubes. PMAG was used to smoothen the grid inside the blocks. The results show better slope continuity at block interface, and improved grid quality.

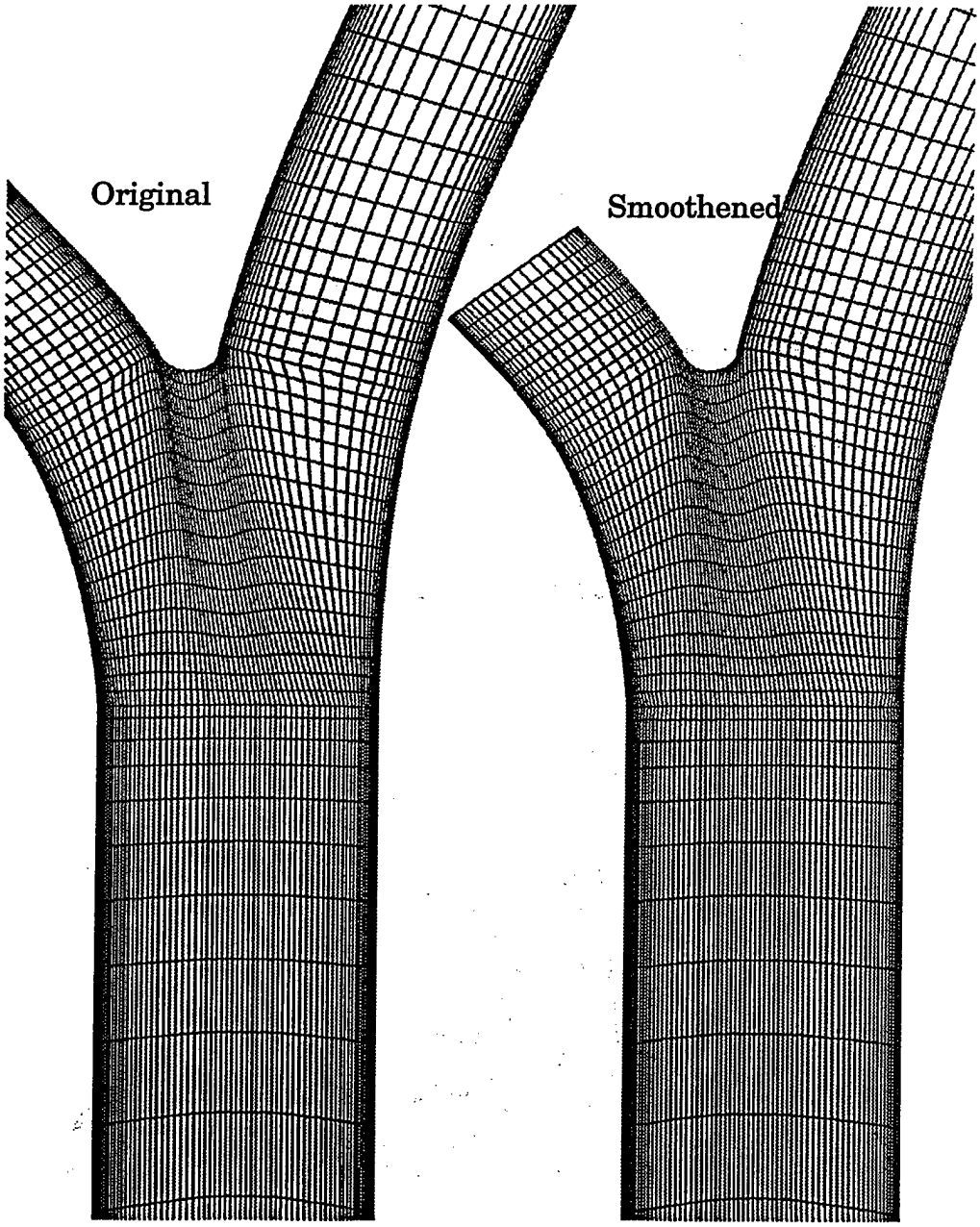


Figure 4.14 Bronchial Tubes

Titan Nozzle

This is a 4 block 2–D geometry of a convergent–divergent nozzle for the titan rocket. SAGE was being used to adapt the grid to this case. However, the grids obtained using SAGE tend to lose the boundary layer packing and give highly skewed grids at the boundaries. Since PMAG is a 3–D code, and requires atleast 3 planes in the k direction a 3–D grid was created by simply stacking the 2–D planes in the k direction. The resulting grid shows excellent shock capturing and a fairly orthogonal grid in all regions. The geometric functions and Neumann boundary conditions implemented in PMAG succesfully produce grids that retain the boundary layer packing as well as orthogonality.

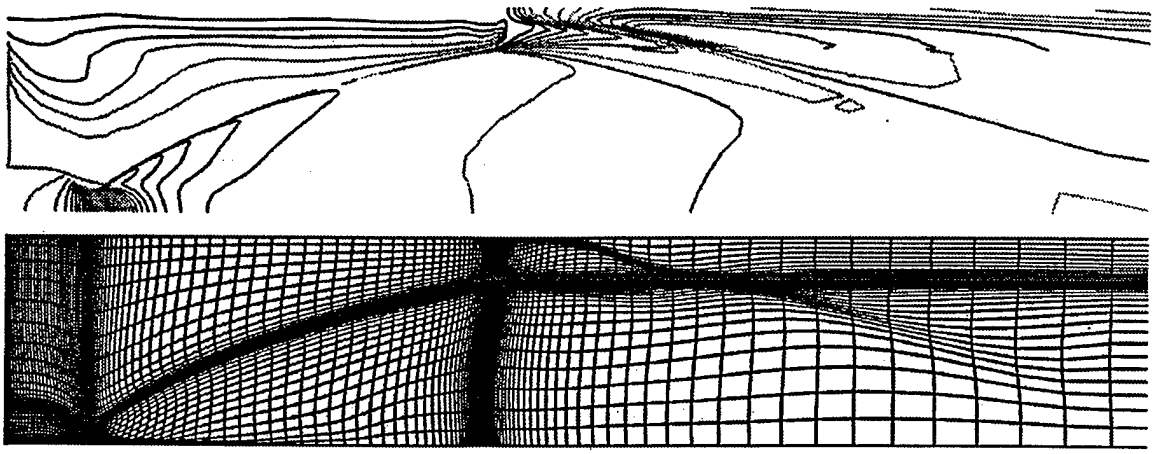


Figure 4.15 Nozzle solution with adapted grid

Conclusions

A parallel grid adaptation algorithm for a general structured multiblock domain (PMAG) has been developed. The algorithm has been demonstrated to handle a range of topologies, like simple multiblock structures, complex structures including multiple blocks sharing a face, faces with fixed as well as movable patches and 0–grids.

Strategies used for interprocess communication of shared faces have proved to work very well. No global connectivity tables are maintained even for general multiblock topologies. Complete continuity of grid lines is guaranteed across shared faces.

Processes can communicate grid point information and solution information after any specified number of iterations or after a specific change in the global norm. This can save on communication overheads, if the grids do not change significantly at every iteration.

Each process operates on one block. Processes exchange messages using the Message Passing Interface (MPI). If the number of processors available is larger than the number of blocks, PMAG can spawn threads to make maximum use of the available resources (Only on SGI shared memory platforms). The speedup achieved has been shown to be significant. However, the efficiency drops since the NURBS and solution interpolation routines do not use multiple threads. In case the number of processors is less than the number of blocks, the processes time–share on the CPUs. This has been shown not to be too inefficient as compared to a single block run.

A weight function generated with a boolean sum of scaled first and second order derivatives of all solution variables has been used. This function

has shown success in identifying solution features of widely varying intensities like shocks, boundary layers, separated regions, and vortices.

Generation of NURBS surfaces using inverse NURBS formulation allows boundary point movement without loss of geometric fidelity. This has helped in generating excellent orthogonal adapted grids.

A multiblock solution interpolation algorithm has been designed. The algorithm has been tested to give very good results. Solution interpolation guarantees grid adaptation in accurate regions.

The algorithm has also been shown to be a useful and flexible tool in generating smooth elliptic grids. The features provided give the user the ability to create structured elliptic grids around very complicated geometries.

This algorithm was designed with the primary objective of speeding up the grid adaptation process. This objective has been met with great success.

Future Work

The areas for further research can be classified as:

Grid Generation Issues

- Weight functions depending on orthogonality and skewness
- Allow user to define variables to be used for calculating weights.
- Extension to hybrid grids.
- Integrate with a flow solver.

Parallel Computing and Performance Issues

- Improve overlap of communication and computation.
- Multiblock solution interpolation using SHIFT.
- Multi–threaded implementation of NURBS routines.
- Implement threads on SUN.

- Automatic load balancing using threads.

General Improvements

- Make the code fool proof with extensive error checking on inputs.

APPENDIX A

TRANSFORMATIONS AND TRIDIAGONAL FORMULATION

Transformation Relations

Grid generation involves transformation from the computational space ξ^i to the physical coordinates x^i. The following definitions help in expressing the metric terms and elliptic equations in a concise form.

Basic Definitions:

$$\underline{a}_i \equiv \underline{r}_{\xi^i} \equiv \left(\frac{\partial x_1}{\partial \xi^i}, \frac{\partial x_2}{\partial \xi^i}, \frac{\partial x_3}{\partial \xi^i}\right), i = 1,2,3 (covariant)$$

$$\underline{a}_i \equiv \nabla \xi^i \equiv \left(\frac{\partial \xi^i}{\partial x_1}, \frac{\partial \xi^i}{\partial x_2}, \frac{\partial \xi^i}{\partial x_3}\right), i = 1,2,3 (contravariant)$$

$$g_{ij} = \underline{a}_i \cdot \underline{a}_j = g_{ji} \qquad \begin{matrix} i = 1,2,3 \\ j = 1,2,3 \end{matrix}$$

$$g^{ij} = \underline{a}^i \cdot \underline{a}^j = g^{ji} \qquad \begin{matrix} i = 1,2,3 \\ j = 1,2,3 \end{matrix}$$

$$g = \det | g_{ij} | = [\, \underline{a}_1 \cdot (\underline{a}_2 \times \underline{a}_3) \,]^2$$

Important Formulae:

$$\sum_{i=1}^{3} (\underline{a}_j \times \underline{a}_k)_\xi = 0 \qquad i,j,k \text{ cyclic}$$

$$(g_{ij})_{\xi^k} = \underline{r}_{\xi^i \xi^k} \cdot \underline{r}_{\xi^j} + \underline{r}_{\xi^i} \cdot \underline{r}_{\xi^j \xi^k}$$

$$\underline{r}_{\xi^i \xi^j} \cdot \underline{r}_{\xi^i} = \frac{(g_{ii})_{\xi^j}}{2}$$

$$\underline{r}_{\xi^i\xi^i} \cdot \underline{r}_{\xi^j} = (g_{ii})_{\xi^j} - \frac{(g_{ii})_{\xi^j}}{2}$$

$$\underline{r}_{\xi^i\xi^j} \cdot \underline{r}_{\xi^k} = \frac{(g_{ik})_{\xi^j} - (g_{ij})_{\xi^k} + (g_{jk})_{\xi^i}}{2} \qquad i \neq j \neq k$$

$$g^{il} = \frac{1}{g} \left(g_{jm}\, g_{kn} - g_{jn}\, g_{km} \right)$$

Gradient

$$\underline{\nabla} A = \frac{1}{\sqrt{g}} \sum_{i=1}^{3} [(\underline{a}_j \times \underline{a}_k) A]_{\xi^i} \qquad \textit{Conservative}$$

$$\underline{\nabla} A = \frac{1}{\sqrt{g}} \sum_{i=1}^{3} [(\underline{a}_j \times \underline{a}_k)] A_{\xi^i} \qquad \textit{Non–Conservative}$$

Laplacian

Conservative

$$\nabla^2 A = \underline{\nabla} \cdot (\underline{\nabla} A)$$

$$= \frac{1}{\sqrt{g}} \sum_{i=1}^{3} \sum_{i=1}^{3} (\underline{a}_j \times \underline{a}_k) \cdot [(\underline{a}_m \times \underline{a}_n) A]_{\xi^i\, \xi^i}$$

Non–Conservative

$$\nabla^2 A = \frac{1}{g} \sum_{i=1}^{3} \sum_{l=1}^{3} (\underline{a}_j \times \underline{a}_k) \cdot (\underline{a}_m \times \underline{a}_n) A_{\xi^i\xi^i}$$

$$+ \frac{1}{\sqrt{g}} \sum_{i=1}^{3} \sum_{l=1}^{3} (\underline{a}_j \times \underline{a}_k) \cdot [\frac{1}{\sqrt{g}} (\underline{a}_m \times \underline{a}_n)]_{\xi^i} A_{\xi^i}$$

Formulation of the Tridiagonal System

Consider the three dimensional elliptic grid generation system :

$$\sum_{i=1}^{3}\sum_{j=1}^{3} g^{ij}\,\vec{r}_{\xi^i\xi^j} + \sum_{k=1}^{3} g^{kk}\,P_k\,\vec{r}_{\xi^k} = 0$$

To solve the above equation numerically. The derivatives at all points inside the field are evaluated using central difference. For points on boundary one sided second order differencing is used.

Central Difference :

$$r_\xi = \frac{r_{i+1} - r_{i-1}}{2} \qquad r_{\xi\xi} = r_{i+1} - 2r_i + r_{i-1}$$

$$r_{\xi\eta} = r_{i+1,\,j+1} - r_{i-1,\,j+1} - r_{i+1,\,j-1} + r_{i-1,\,j-1}$$

Forward difference :

$$r_\xi = \frac{-3\ r_i + 4\ r_{i+1} - r_{i+2}}{2}$$

Backward difference :

$$r_\xi = \frac{3\ r_i - 4\ r_{i-1} + r_{i-2}}{2}$$

The equation is then solved using a tridiagonal solver. If r_i *(i=1,n)* be the points along an i line, the entire line can be evaluated by casting the equations as a set of linear equations as follows:

$$\begin{bmatrix} d_1^2 & d_1^3 & 0 & 0 \\ d_2^1 & d_2^2 & d_2^3 & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & d_{n-1}^1 & d_{n-1}^2 & d_{n-1}^1 \\ 0 & 0 & d_n^1 & d_n^2 \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \\ \cdots \\ r_{n-1} \\ r_n \end{bmatrix} = \begin{bmatrix} R_1 \\ R_2 \\ \cdots \\ R_{n-1} \\ R_n \end{bmatrix}$$

where,

$$d_i^1 = g^{11} \left(1 - 0.5 \; P_1 \right)$$

$$d_i^2 = -2 \left(g^{11} + g^{22} + g^{33} \right)$$

$$d_i^3 = g^{11} \left(1 + 0.5 \; P_1 \right)$$

$$\begin{aligned} R_i = & -2 \left(g^{12} \; r_{\xi\eta} + g^{23} \; r_{\eta\zeta} + g^{31} \; r_{\zeta\xi} \right) \\ & - g^{22} \left(r_{i,j+1,k} + r_{i,j-1,k} + P_2 \; r_\eta \right) \\ & - g^{33} \left(r_{i,j,k+1} + r_{i,j,k-1} + P_3 \; r_\zeta \right) \end{aligned}$$

This system of linear equations is solved for each i line.

APPENDIX B

USERS GUIDE

Introduction

PMAG is a 3–dimensional Parallel Multiblock Solution–Adaptive Grid generator based on solution of elliptic partial differential equations. It can also be used for generating smooth orthogonal grids on complex multiblock domains. The following sections guide the user through the steps required for using this code. The user is assumed to be familiar with basic grid generation and the numerical flow simulation process.

Overview and Terminology

PMAG is a structured grid adaptation algorithm. Adaptation is done by *redistribution* of grid points. The total number of grid points remains the same.

A block has 6 faces, 12 edges and 8 vertices. There is no restriction on block – block connectivity. One block can be share its faces, edges or vertices with any number of blocks. The numbering of blocks is not important. Any block can be linked to any other block including itself. The part of the block face that is shared with another block/ held fixed / defined as a NURBS surface is called a *patch.* Each face can have MAXPATCH number of patches. *Shared patches* are free to move in space. *Fixed Patches* are held fixed throughout the adaptation process. These points do not move. *NURBS patches* are defined on the surfaces where the geometric fidelity needs to be conserved. The grid points on these surfaces are moved such that they do not distort the original geometry. *Block numbers start with 0.*

Each block is solved in an individual process. The process communicate with each other using MPI. The processes can be on the same machine or on different machines. MPI is also supposed to support a heterogenous network. (However, the tests so far have not demonstrated this feature). Machine

names and executable file names for *each* block are specified in the *PMAG.pg* process group file.

On SGI machines, each process can spawn *light weight processes* called *threads.* The process assigns a part of its block to each of its threads. Each thread can run on an individual CPU, thus taking advantage of all the available processing power. The user has the option to specify the CPU on which each process and its associated threads are run. (See local input files)

The Basic Steps

These are the basic steps to generating/adapting grids

- Edit *PARAM.INC* and compile PMAG.
- Put each block of the grid into files *block.0, block.1 .. block.n*
- If adapting, each solution block must be in *soln.0, soln.1 .. soln.n*
- Create local information files *info.0, info.1 .. info.n* for each block.
- Create a global information file *info.global*
- Create the process group file *PMAG.pg*
- Run PMAG.

The output files are:

gridout.0, gridout.1 ... gridout.n : Ascii Grid Blocks in Plot3d format.

soln.0, soln.1 ... soln.n : Ascii Solution Blocks in Plot3d format.

The Include File "PARAM.INC"

PMAG is almost completely in FORTRAN. The PARAM.INC file contains the maximum array sizes and related parameters that need to be set *before* compiling.

ITMAX:	Maximum number of Iterations to run.
MAXi: MAXj: MAXk:	Set these to atleast 2 values larger than the maximum dimension in the respective direction. This is to accomodate the ghostfaces for face exchange.

MAXd: Set to the largest of MAXi, MAXj, MAXk

MAXBLOCKS: The maximum number of blocks in the grid.

MAXPATCH: Maximum number of Patches on the block faces.

MAXCPUS: Maximum number of CPUs on the machine.

MAXOUT: Maximum number of points that may move outside a blocks domain.

NOSLOPE: = 1 boundary point movement criteria is orthogonality.
= 0 criteria is slope continuity for lines near surface.

The user is advised not to change any of the other defined parameters.

The Input files

Solution and grid is expected in PLOT–3D formatted multiblock format. PMAG expects an input of a 3–D multiblock grid in multiple files. Each file contains one block. (This means that each file will have NZONES = 1. This is required in the view of future compatibility). The blocks should be in files block.0 to block.*n* where *n* is the number of blocks in the grid. The corresponding solutions should be in soln.0 to soln.*n*.

There are three main input files.

info.## : These are local input files. ## corresponds to the block number to which this information applies.

info.global: This is the global information file. This information is read only by the 1st block and then broadcast to all.

PMAG.pg: This file is required by MPI to start up processes for each block.

Local input files: *info.##*

The local input file allows the user to specify the shared, fixed and NURBS patches on the block. It also allows the user the specify if the block has to be run on multiple threads (SGI only). The user can also specify a list of CPU's on which the process and its threads are run. If Number of Threads is

specified as 0, the process spawns the default number of threads specified in the global information file. Specify CPU's = 0 leaves the scheduling to the operating system.

```
3                                   Number of THREADS
1                                   Specify CPUs ? (1/0)
  0 1 3 4                           List of CPUs if line 2 = 1
0                                   Imin has 0 shared patches.
3                                   Imax 2 has 3 shared patches
  1,1,29,1,64                       block,jmin,jmax,kmin,kmax
   2,41,71,1,64
   3,1,70,62,97
0                                   Jmin has 0 patches
0                                   Jmax has 0 patches
0                                   Kmin has 0 patches
0                                   Kmax has 0 patches
1                                   Any Fixed Patches? (1/0)
1                                   0 Fixed Patches on Imin
 1,1,70,1,97                        NURBS?,jmin,jmax,kmin,kmax
1                                   1 Fixed Patch on Imax
 0,30,40,1,63                       NURBS?,jmin,jmax,kmin,kmax
0                                   0 Fixed Patches on Jmin
0                                   0 Fixed Patches on Jmax
0                                   0 Fixed Patches on Kmin
0                                   0 Fixed Patches on Kmax

NOTE: Faces must be specified in cyclic order. for example
a J=constant face is specified as kmin,kmax,imin,imax
```

Sample Local Information File

A *NURBS patch* is specified in the section for fixed patches, since the surface geometry is held fixed. The boundary points however are allowed to move on the surface. Fixed patches are also specified with 5 numbers. The first integer specifies whether the patch is fixed (0) or a NURBS (1).

By default each boundary face is treated as a fixed Dirichlet boundary condition. A shared patch is defined by 5 numbers. Neighbouring Block, minimum and maximum index in I direction, minimum and maximum index in J direction. The indices must be in a cyclic order. That is for a J Face, the patch will be specified as : *NbrBlock, Kmin, Kmax, Imin, Imax.*

Global input file: *info.global*

Norm of change in grid point locations is calculated globally at the end of every iteration. The first parameter defines the convergence criteria for the grid. The program stops if the norm is not reached within the Max Number of iterations (2nd line).

If the original grid is extremely distorted and has crossovers etc, its best to start the smoothening process with a few laplace iterations to get rid of the crossovers.

A 0 for RUNTHREADS parameter in the local input file spawns the default number of threads specified in this file.

Geometric Interpolation functions can either be interpolated from the boundaries (1) or calculated over the entire grid (0)

If the starting grid consists of faces only, then the algorithm can generate a starting guess Linear Transfinite Interpolated grid in the volume.

If Adaptation = 0, the algorithm will not read a solution file, and just run as a elliptic grid smoothener.

Sharp changes in weight function can lead to sharp adaptive control functions. These can lead to grid crossovers. Its hence best to smoothen the adaptive control functions by simple averaging. Each iteration extends the field of influence of the control by one grid point in all directions.

```
Maximum norm of change to stop iterations
1.0D-7
Max number of iterations (should be <=ITMAX in PARAM.INC)
35
Number of Laplace Iteration at the beginning.
0
Default Number of threads on large blocks
1
Interpolate geometric control functions? (1/0)
0
Generate Transfinite interpolated grid from faces? (1/0)
0
Adaptation (1/0)
0
Smoothening iterations on the Adaptive control functions
2
Use boolean weights?(1/0)
1
A small value epsilon for normalizing derivatives
0.00001
Multiplication factors for Geometric Control.P,Q,S
1.5 3.5 0.0
Multiplication factors for Adaptive Control P,Q,S
3.5 3.5 0.0
NURBS patches update after how many iterations ?
6
Redistribute solution on new grid? (1/0)
0
Redistribute after every # iterations( If Redistribute=1)
10
```

Sample Global Input File

Boolean Weights = 1 uses a weight function obtained by a boolean sum of weights in each direction. Else adaptive control functions are calculated in each direction only with respect to the weight function in that direction.

Epsilon serves as a small number to avoid a division by zero. It also acts as a filter for the flow features resolved.

The multipliers magnify the control functions by the specified amount. This way the user can change the effectiveness of each of the factors controlling the grids.

NURBS patches can be updated every few iterations. Updating the patches every 5–6 iterations is usually good enough, but it depends on application and how much the grid changes with every iteration.

The last parameter specifies the number of iterations between solution interpolation. Again 3–5 iterations is usually a good number, but the requirements can vary for each case.

The process group file: *PMAG.pg*

This file lists the machines where each process will be created.

MPI spawns the process by running rshell.

```
local          0       /tmp/manoj/PMAG
machine1       2       /usr/tmp/manoj/PMAG
machine2       1       /tmp/PMAG
machine2       1       /tmp/PMAG
```

Sample Process Group File

Except for the first line, each line consists of

Machine–name Number_of_processes Full_path_to_executable.

More than one processes can be spawned on one machine in either of the ways as shown in the example file.

The first line consists of

local 0 Full_path_to_executable.

The first line spawns the root process on the local machine.

The number of processes specified in this file MUST be equal to the total number of blocks! For more information on the process group file refer to the MPI reference manual.[27]

REFERENCES

[1] Soni, B.K., "Grid Generation: Algebraic and Partial Differential Equations Techniques Revisited", Computational Fluid Dynamics '92, Vol.2, pp. 929–937, Elsevier Science Publishers, 1992.

[2] Thornburgh, H. and Soni B.K., "A Structured Grid Based Solution–Adaptive Technique for Complex Separated Flows", To appear in *Journal of Applied Mathematics and Computations.* (Accepted 9/2/1996)

[3] Thornburgh H.J. and Soni B.K., "Weight Functions in Grid Adaption," *Proceedings of the 4th International Conference in Numerical Grid Generation in Computational Fluid Dynamics and Related Fields.* Swansea, Wales, April 1994.

[4] Ghia, K.N., Ghia, U., Shin, C.T. and Reddy, D.R., "Multigrid Simulation of Asymptotic Curved–Duct Flows Using a Semi–Implicit Numerical Technique," Computers in Flow Prediction and Fluid Dynamics Experiments, ASME Publication, New York 1981.

[5] Thompson, J.F., "Composite Grid Generation for General 3–D Regions–the EAGLE Code", *AIAA Journal,* Vol 26, No. 3, pp. 271, 1988.

[6] Kim, H.J. and Thompson, J.F., "Three Dimensional Adaptive Grid Generation on a Composite–Block Grid", *AIAA Journal,* Vol 28, No. 3, 1990.

[7] Thompson J.F. and Gatlin, B., Program EAGLE User's Manual, Volume III – Grid Generation Code, AFATL–TR–88–117, Eglin AFB, 1988.

[8] Nakahashi, K. and Deiwert, G.S., "A Self Adaptive Grid Method with Application to Airfoil Flow", AIAA Seventh Computational Fluid Dynamic Conference, Cincinnati, Ohio, July 1985. (AIAA Paper 85–1525).

[9] Davies, C.B. and Venkatapathy, E., "The Multidimensional Self–Adaptive Grid Code, SAGE", NASA TM–103905, July 1992.

[10] Kumar V., et. al., Introduction to Parallel Computing : Design and Analysis of Parallel Algorithms.,The Benjamin/Cummings Publishing Company Inc., Redwood City, 1994.

[11] Thompson J., et.al., Numerical Grid Generation : Foundation and Applications., North–Holland, 1985.

[12] Eiseman, P.R., "Adaptive Grid Generation", *Computer Methods in Applied Mechanics and Engineering,* Vol. 64.231, 1987.

[13] Anderson, D.A., "Adaptive Grid Methods for Partial Differential Equations", *Advances in Grid Generation,* ASME Fluids Engineering Conference, Houston, 1983

[14] Eiseman, P.R, "Alternating Direction Adaptive Grid Generation", AIAA Paper 83–1937, 1983

[15] Thompson J.F., "A Survey of Dynamically–Adaptive Grids in the Numerical Solution of Paritial Differential Equations", *Applied Numerical Mathematics,* Vol 1, pp 3, 1985.

[16] Yang J.C., "General Purpose Adaptive Grid Generation System",Ph.D. Dissertation, Mississippi State University, 1993.

[17] Soni B.K. and Yang, J.C.,"General Purpose Adaptive Grid Generation System", AIAA–92–0664, 30th Aerospace Sciences Meeting, Reno, NV, January 1992.

[18] NASA LeRC and USAF AEDC, NPARC 1.2 User Notes, June 1994.

[19] Mastin C.W., "Fast Interpolation Schemes for Moving Grids", Proceedings of the Second International Numerical Grid Generation Conference ,Miami, December 1988.

[20] Stokes, M.L. and Kneile, K.R., "A Three–Dimensional Search/Interpolation Scheme for CFD Analysis",Presented at the *First World Congress on Computational Mechanics*, University of Texas at Austin, September 1986.

[21] Stokes, M.L. "Streamline Equation Integration for Visualizing Time Dependent Computational Field Simulations," *Proceedings, SECTAM XVI Conference,* University of Tennessee Space Institute Nashville, TN, April 1992.

[22] Farin G., Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide, Academic Press, 1992

[23] Khamayseh A., "Elliptic Surface Grid Generation on Analytically Defined Geometries", Ph.D. Dissertation, Mississippi State University, May 1994.

[24] Yu, T.Y., "CAGD Techniques in Grid Generation", Ph.D. Dissertation, Mississippi State University,

[25] Foster Ian, Designing and Building Parallel Programs, Addison–Wesley, 1995.

[26] Gropp, W., Lusk, E. and Skjellum, A., Using MPI: Portable Parallel Programing with the Message–Passing Interface, MIT Press, 1994

[27] Snir, M., Otto, S.W., Huss–Lederman, S., Walker, D.W. and Dongarra, J., MPI: The Complete Reference, MIT Press, 1996.

[28] Arney, D.C. and Flaherty, J.E., "An Adaptive Mesh–Moving and Local Refinement Method for Time–Dependent Partial Differential Equations,"*ACM Transaction on Mathematical Software*, Vol. 16, No. 1, pp. 48–71, March 1990.

[29] Kohn S., Weare J., Ong E. and Baden S., "Parallel Adaptive Mesh Refinement for Electronic Structure Calculations," Eighth SIAM Conference on Parallel Processing for Scientific Computing, Minneapolis, March 1997.

[30] Sturek, W.B., Birch, T., Lauzon, M., Clinton, H., Manter, J., Josyula, E. and Soni, B.K., "The Application of CFD to the Predicttion of Missile Body Vortices," AIAA 96–0637, January 1997.

BIBLIOGRAPHY

1. SHIH, M.H., "Towards a Comprehensive Computational Simulation System for Turbomachinery," Ph.D. Dissertation, Mississippi State University, May 1994.
2. SONI, B.K., THOMPSON, J.F., STOKES, M.L. and SHIH, M.H., "GENIE++, EAGLEVIEW, and TIGER: General Purpose and Special Purpose Graphically Interactive Grid Systems," AIAA–92–0071, *AIAA 30th Aerospace Sciences Meeting,* Reno, NV, January 1992.
3. CLARKE, J., "Network Distributed Global Memory for Transparent Message Passing on Distributed Networks," Army Research Laboratory Technical Report, 1994.
4. PATEL, N., "DZONAL"
5. THOMPSON, J.F., "The National Grid Project," *Computer Systems in Engineering,* Vol. 3, Nos. 1–4, pp. 393–399, 1992.
6. BEACH, T.A., "An Interactive Grid Generation Procedure for Axial and Radial Flow Turbomachinery," AIAA–90–0344, *AIAA 28th Aerospace Sciences Meeting,* Reno, NV, January 1990.
7. YU, T.Y., "IGES Transformer and NURBS in Grid Generation," Master's Thesis, Mississippi State University, August 1992.
8. SHIH, M.H., YU, T.Y. and SONI, B.K., "Interactive Grid Generation and NURBS Applications," Accepted for publication , *Journal of Applied Mathematics and Computations.*
9. FARIN, G., "Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide," Third Edition, Academic Press, 1990.
10. Deboor, C., "A Practical Guide to Splines," Springer–Verlag, New York, 1978.
11. MIDDLECOFF, J.F. and THOMAS, P.D., "Direct Control of the Grid Point Distribution in Meshes Generated by Elliptic Equations," AIAA–79–1462, *AIAA 4th Computational Dynamics Conference,* Williamsburg, VA, 1979.
12. COOPER, G.K. and SIRBOUGH, J.R., "PARC Code: Theory and Usage," AEDC–TR–89–15, Arnold Engineering Development Center, Arnold AFB, TN, 1989.
13. GEIST, A., et. al., "PVM 3 Users's Guide and Reference Manual," ORNL/TM–12187, Oak Ridge National Laboratory, TN, 1994.
14. "MPI: A Message–Passing Interface Standard," *Message–Passing Interface Forum,* May 1994.
15. NASA LeRC and USAF AEDC, "NPARC 1.0 User Notes," June 1993.
16. NASA LaRC, "User Document for CFL3D/CFL3DE (Version 1.0)," 1993.
17. YANG, J.C. and SONI, B.K., "Structured Adaptive Grid Generation," *Proceedings of the Mississippi State University Annual Conference on Differential Equations and Computational Simulation,* Mississippi State University, Mississippi State, MS, March 1993.
18. THORNBURG, H.J., "An Adaptive Grid Technique for Simulation of Complex Unsteady Flow," Ph.D. Dissertation, University of Cincinnati, Cincinnati, OH.

APPENDIX

A.1 TIGER SYSTEM

The development of TIGER has evolved from its original grid generation purpose into a comprehensive simulation system that contains six modules: (i) grid generation module, (ii) visualization modules, (iii) network module, (iv) flow solution module, (v) simulation module, and (vi) toolbox module. Each may be accessed independently to perform different tasks. These modules are linked together with a common graphical user interface (GUI) [Figure 1].

Advances in graphics workstations have provided the hardware capability for the execution of sophisticated, user–friendly and interactive grid generation and scientific visualization softwares. A rising awareness on the issues such as data structures and grid generation algorithms prompts this development which has dramatically reduced the man–hours involved during the grid generation process for many engineering applications. Computer softwares such as GENIE++ [2], EAGLEView[2], and NGP[5] are examples of such developments. Basic grid generation techniques, such as transfinite interpolation (TFI), weighted TFI (WTFI), elliptic grid generation systems, etc., are refined during the development of these codes. However, for applications involving complex turbomachinery flow fields, the grid generation process is extremely time consuming. Consequently, customized grid generation codes for turbomachinery configurations have been developed to address this problem[3,4,6].

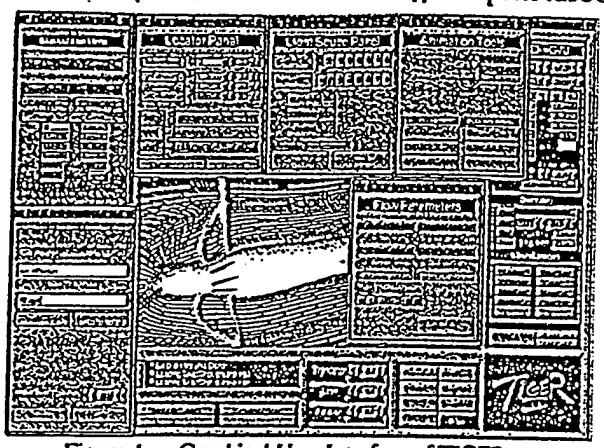


Figure 1 Graphical User Interface of TIGER

In TIGER, the grid generation step is accomplished by the combination of automatic and interactive customized algorithms including computational domain transformation, geometry surface construction, sub–domain frame setup, and local grid manipulation. The domain transformation is automatically performed by mapping the physical domain into computational domain once the user specifies the indices for the blade and block boundaries. This index information is also used to perform automatic sub–blocking during the later grid generation process. An interactive procedure is designed to allow the user to manipulate critical η–constant curves, such as the grid line that run through the rotor tip or the shroud, where η is the curvilinear coordinate in radial direction. The major component of a turbomachine, the blade, is constructed automatically by performing intersection and surface spline–fit with non–uniform rational B–spline (NURBS) algorithm [3,7,8,9].

NURBS has drawn a lot of attention in the areas of geometry modeling, computer graphics, and grid generation because of its powerful geometry properties such as the convex hull, local control, variation diminishing and affine invariance. It also has useful geometry tools such as knot insertion, degree elevation, and splitting[7,8,9].

A NURBS curve of $m+1$ control points can be defined by an order k, a set of control points $\{b_i,\ i=0,..,m\}$, a set of knots $\{u_i,\ i=0,..,m+k\}$, and a set of weights $\{w_i,\ i=0,..,m\}$. The following equations represent a NURBS curve:

$$c(u) = \frac{\sum_{i=0}^{m} w_i\, b_i\, N_i^k(u)}{\sum_{i=0}^{m} w_i\, N_i^k(u)} \qquad (1)$$

where the basis functions N_i^k are defined as

$$N_i^k(u) = \frac{(u - u_i)N_i^{k-1}(u)}{u_{i+k-1} - u_i} + \frac{(u_{i+k} - u)N_{i+1}^{k-1}(u)}{u_{i+k} - u_{i+1}}$$

$$N_i^1(u) = 1 \text{ if } u_i \le u < u_{i+1}$$
$$= 0 \qquad i = 0, 1, \ldots, m \qquad (2)$$

A NURBS surface with $(m+1)\times(n+1)$ control points can be expressed by two orders k_u and k_v, a set of control points $\{b_{ij}, i=0,..,m, j=0,..,n\}$, a set of knots $\{(u_i), i=0,..,m+k_u, (v_j), j=0,..,n+k_v\}$, and a set of weights $\{w_{ij}, i=0,..,m, j=0,..,n\}$. The following equation represents the NURBS surface:

$$s(u,v) = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} w_{ij}\, b_{ij}\, N_i^{k_u}(u)\, N_j^{k_v}(v)}{\sum_{i=0}^{m}\sum_{j=0}^{n} w_{ij}\, N_i^{k_u}(u)\, N_j^{k_v}(v)} \qquad (3)$$

where the basis functions $(N_i^{k_u}, N_j^{k_v})$ are defined in a similar fashion as Equation (2). The De Boor algorithm[10] can be used to evaluate the NURBS curves and the NURBS surfaces with known evaluation order(s), control points, weights, knot sequences, and hence the basis functions.

The entire domain is represented in a framework by connecting each critical index location, such as the blade leading edge, trailing edge, and hub nose, etc.. Both meridional view and cascade view are available to a user for specifying the distribution informations, and allow the manipulations to these frames. An example for the cascade frame, meridional frame, and the corresponding surface grid of a high bypass ratio fan stage are shown in Figure 2.

The meridional frame represents the entire domain on (x,r) plane, where (x,r,θ) are the cylindrical coordinates used in TIGER. The cascade frame represents the blade-to-blade surface in parametric coordinates (m',σ). The transformation relationship between (x,r,θ) and (m',σ) can be expressed as follows [5,11]: for a generatrix g=g($\hat{x}(u)$,$\hat{r}(u)$)

$$\bar{m}'_i = \int_{u_0}^{u_i} g(\hat{x}(u),\hat{r}(u))\, du$$

$$\sigma_{ij} = \bar{r}\theta_{ij}$$

where

$$\bar{r} = \left\{\sum_{i=0}^{n_i} \hat{r}_i\right\}/(n_i + 1)$$

$$\bar{m}'_0 = 0.0$$

$$i = 0, \ldots, n_i$$

$$j = 0, \ldots, n_j$$

Each point $p(x_{ij},r_{ij},\theta_{ij})$ on the associated surface of revolution is compared with the associated $\bar{m}'$–x or $\bar{m}'$–r map to obtain the proper m'–coordinate. Example maps for an industrial ventilation fan presented

in Figure 3 are shown in Figure 4. However, if a hub generatrix is not a single–valued function, such as the example shown in Figure 4(a), there is a risk of losing the one–to–one correspondence. Therefore, a strategy is taken to safeguard this critical one–to–one correspondence by decomposing the generatrix into various sections based on the slope information. Arrows in Figure 4 represent the section break positions. For a generatrix section, if the slope $|s| < 1.0$, $\overline{m}'$–x map is used, otherwise $\overline{m}'$–r map is utilized. Therefore, for each point $p(x_{ij}, r_{ij}, \theta_{ij})$, m'_{ij} is obtained by comparing with either $\overline{m}'$–x map or $\overline{m}'$–r map. The associated transformed cascade frame and the corresponding grid are shown in Figure 5.

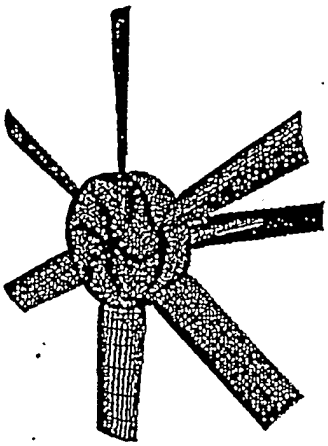
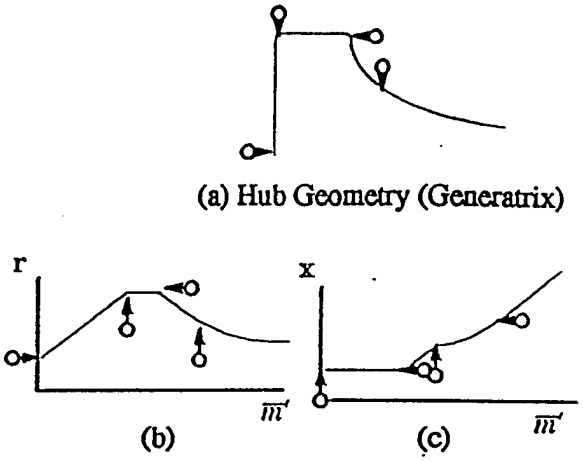


Figure 3 Numerical Model of an Industrial Ventilation Fan



(a) Hub Geometry (Generatrix)

(b) (c)

Figure 4 Hub Geometry and the Associated Maps for an Industrial Fan

(a) Transformed 2D Surface (b) Resulting 3D Surface

Figure 5 Cascade Surface Mapping (Hub)

The cascade surface mapping scheme developed in this work is designed to spline–fit each point from (m', σ) space to desired surface of revolution point by point. It is therefore possible to allow surface grid generation with different topology.

For a blade with low stagger angle and large blade leading edge (LE) circle radius, it is a very difficult region to maintain non–overlapping grid with algebraic approach near the leading edge. Examples of this problem are demonstrated in Figure 7(a). A common practice to resolve this problem is to apply elliptic system with proper control functions, such as the Thomas–Middlecoff type control functions[11]. Results of elliptic smoothing with Thomas–Middlecoff–type control function are presented in Figure 6(b). In this figure, it is clearly demonstrated that the elliptic smoothing eliminates the over–lapping grid in

the problem region. However, as demonstrated in the close–up regions, the grid spacing is being pulled off the concave region and being pushed closer to the convex boundary, creating a sudden change in grid spacing, as shown in Figure 6(b). It is induced by performing surface elliptic smoothing patch by patch.

In order to improve the grid quality, one more step is taken in the algorithm of TIGER's grid generation module. That is, the surface with elliptic smoothing is spline–fitted with NURBS surface formulation. Elliptic smoothing is necessary because the NURBS formulation requires a non–overlapping initial grid to begin with. As demonstrated in Figure 6(c), the grid quality has been improved. It is shown in Figure 8 that for this case the orthogonality near the blade surface is improved as well.

The grid generation module of the TIGER system is one of the most significant features that reduces the labor requirements for CFD applications associated with axisymmetric flow fields. This module has been used to construct structured grids for axial, radial, and mixed–flow axisymmetric configurations. Examples for internal flow field applications are the high Reynolds number pump as shown in Figure 7, the NASA low speed centrifugal compressor as presented in Figure 8, and missile configuration as an example for an external flow field application, as shown in Figure 9.

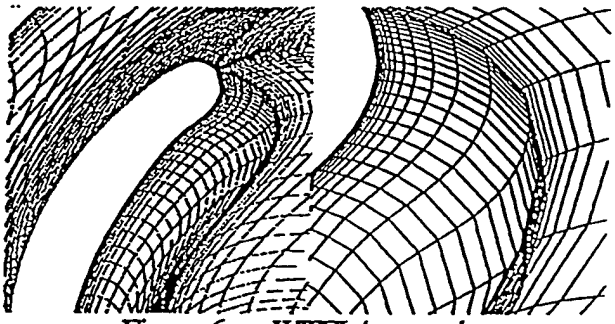


Figure 6a WTFI Approach

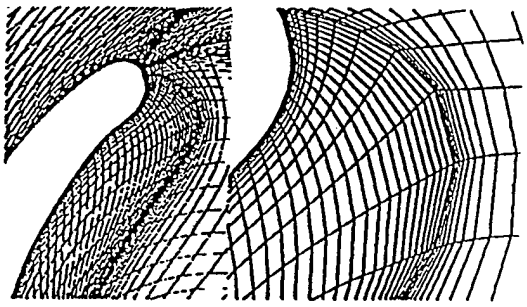


Figure 6b Elliptic Approach

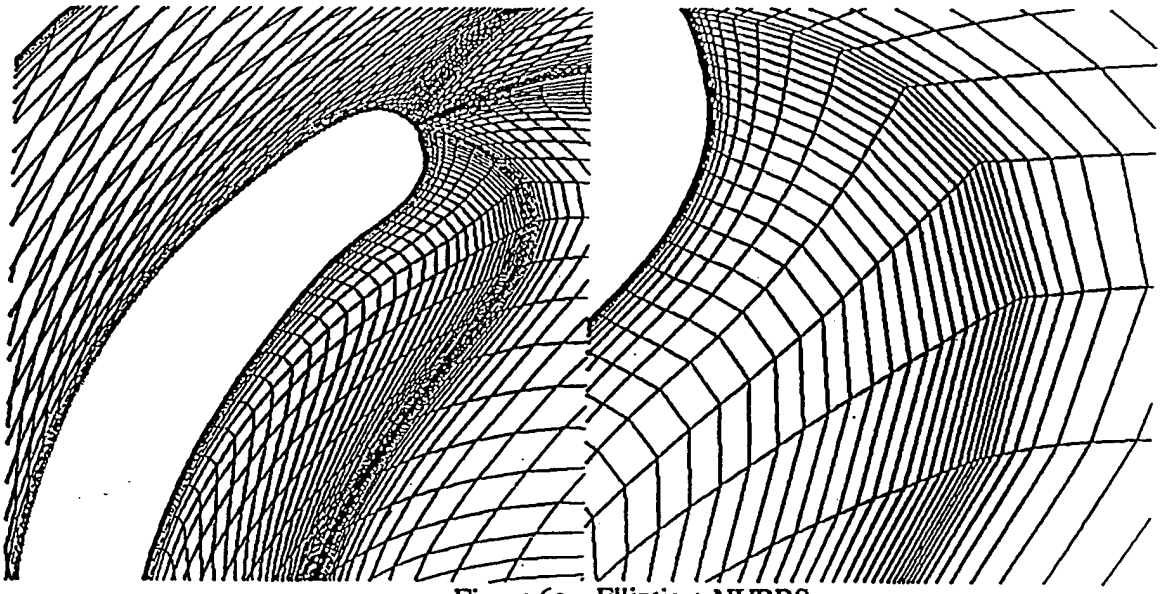


Figure 6c Elliptic + NURBS

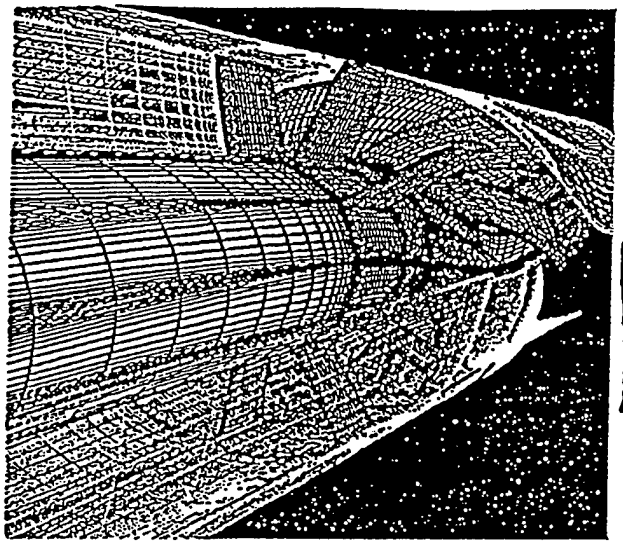


Figure 7 High Reynolds Number Pump

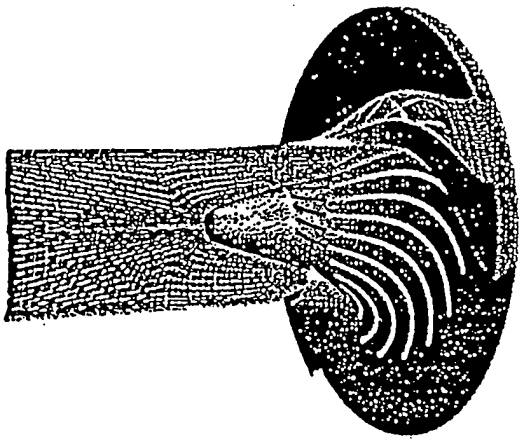


Figure 8 Low Speed Centrifugal Compressor

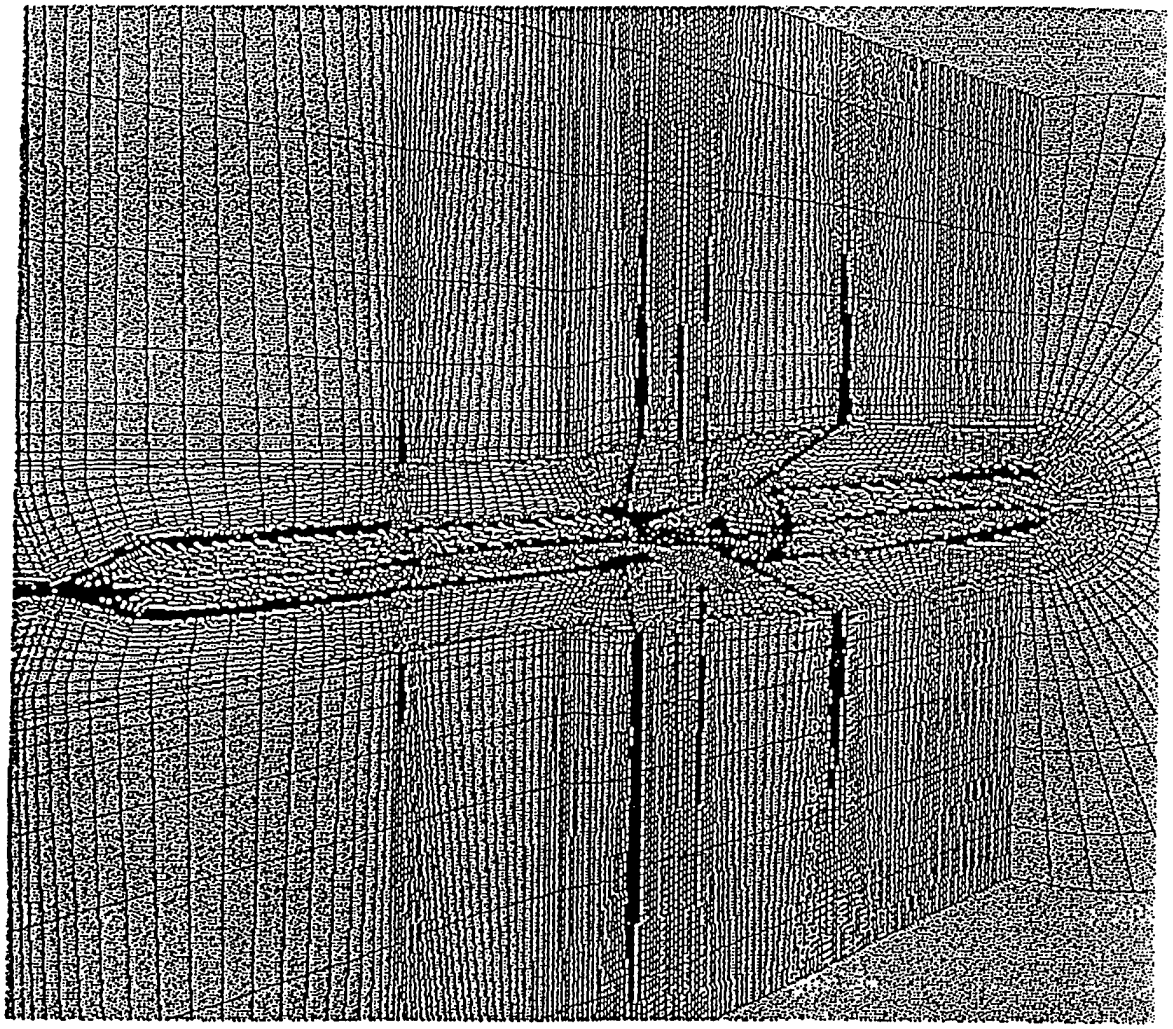


Figure 9 Missile Configuration